# XSP: An IntegrationTechnology for Systems Development and Evolution
## *Formal Specifications for Unifying XML and Relational Systems*

Michael Champion *Software AG*[*]
July 12, 2001

## ***Abstract***

Before the "relational revolution", developers had to understand the details of the data structures on which they operated in order to access the content. Relational theory introduced the concept of "data independence" in that the operations in the relational algebra do not need to know anything about the structure of the data in order to work. This allows analysts, designers, and programmers to do their jobs without knowledge of the structure the underlying physical data, and for database administrators to reorganize, optimize, and distribute physical databases without "breaking" applications

The mathematics of the relational model is based heavily on classical set theory, CST, and this is both its strength and its weakness. For example, limitations -- such as the ability to talk meaningfully only about flat tables -- stem from the fact that classical set theory blurs the distinction between sets with "ordered" elements and sets with "nested" elements. Thus, operations become ill defined when extended to represent and manipulate sets with both ordered elements and nested elements. D L Childs developed an "extended set theory", XST, more than 30 years ago that adds an additional parameter known as a "scope" to the membership condition of classical set theory. In CST, membership is based on only an element component; in XST membership is based on both an element component and a scope component. This extended membership condition can be used to model ordering and containment relationships that are simply too "messy" to handle in classical set theory and the formalisms (such as relational algebra) that are based on it.

XSP Technology consists of three separate formal specifications:

> **XST: Extended Set Theory -** Formal axiomatic specification of extensions to the foundations of Classical set theory that support the modeling of computer based operations and operands**.**
> **XSN: Extended Set Notation -** Formal set theoretic notation expressing operations and operands of XST that preserve the mathematical identity of all conceptual and computer-based structures being considered for a system**.**
> **XSP: Extended Set Processing** - Formal specification of a system of XSN defined operations and operands that can be executed by a computer.

In applying XSP technology to real-world problems, the "well-formedness" criterion of XML means that XML data can be formally modeled by extended set theory (XST). In other words, an XML "document" is, by its very nature, a well-defined extended set. Previous XSP research has shown the feasibility of preserving the operation-centric and data independent advantages of the relational data model (RDM) by formally modeling XML document types as extended sets (Xsets).

XSP Technology also goes beyond the formal underpinnings of the relational model to describe the physical implementation of the operations and the representations of data within a computer. This has great potential to bridge the mathematically clean, but difficult world of the "pure" relational model and the messy, but more pragmatic world of XML. As the concepts of XSP technology become more widely known, they can be applied and expanded by widespread experience and experimentation. XSP technology offers the advantages of mathematical formalism to be applied to the XML world of structured documents.

[*] Software AG supported this analysis of XSP Technology, but that company does not necessarily endorse the conclusions here, nor should any implications be drawn as to the company's plans to support XSP Technology in its future products.

# Introduction

The greatest challenge in developing any computer-based system is in communicating the functional objectives that are understood by humans into the precision mechanism understood by computers. The real world of people, with its intrinsic ambiguity of expression, depends on the understood meaning of potentially ambiguous messages. The internal world of the computer, with its relentless precision of execution, is very hostile toward ambiguity. Unfortunately, the understood meaning that is apparent to humans is often “in the eye of the beholder.” Seldom, if ever, is the computer aware of which beholder understands which meaning.

Thus, the greatest software development challenge stems from the need to map the ambiguous views of people into an internal representation that a computer can work on. This problem of defining the view/internal interface has been investigated from many angles since the 1960’s, yet there is still no widely accepted formal method for describing the VIEW environment (as seen by the user), the INTERNAL environment (of the computer), and the mapping between the two. All that has been established over the past thirty years is that none of the established formal disciplines is up to the task. This includes classical set theory, category theory, lambda calculus, graph theory, the relational model of data, and even denotational semantics. Many modeling attempts have been made, but they all have failed for one reason: no mathematical discipline exists that is rich enough to model the mathematical identity of objects as perceived by humans, as represented in a machine, and as programs that provide function preserving transformations between the two.

It is important to emphasize the necessity for a formal, mathematical model of the two environments and the transformation between them. Certainly there are millions of humans who can use the formalisms of category theory, relational algebra, or whatever to understand a problem in the VIEW environment and then write programs that (at least appear to) implement a solution in the INTERNAL environment. The critical difference is that the mapping between the formalisms of the VIEW and the implementation in the computer’s INTERNAL environment requires *human intelligence* to resolve the various ambiguities. While we think of mathematics as requiring intelligence to perform, in fact the mathematical formalisms themselves are “dumb” – once defined, they operate in a relentless, mechanical manner that depends only on explicitly defined operations and operands, not on implicit “knowledge.” The implicit knowledge required for human understanding often gets in the way of effective translation into the internal environment of the computer. Without a formal modeling foundation, no formal VIEW/INTERNAL interface technology can be developed for communication the functional objectives of the human world to the precision mechanism of the computer world.

## *XML CHALLENGES*

Nowhere is the need for such a formal modeling foundation more evident than it is for the emerging technology to develop systems based on XML structures. The “dark side” of any formal modeling system is generally the use of arcane mathematical notation. (XSP technology is no exception!) The “bright side” is that all the mathematics can be ignored, eventually. Its existence is of most importance, but the actual mathematics can be buried under friendly layers of abstraction. The real purpose of the mathematics is to *absorb complexity* and to capture minute details with excruciating precision. These notational handles are the key ingredient to any algebraic, or operation-centric, modeling environment. A simple example should set the stage for appreciating why XML structures prose a significant challenge to being captured by any established formal modeling mechanism.

The sentence “Alan is 42 years old and his e-mail address is agb@abc.com” expresses a relationship between the Name (n) of a Person (p), the Age (a) of the same Person (p), and the E-mail address of the same Person (p).

The most formal notation is classical set theory, where

```
p = { <a, 42>, <n, Alan>, <e, agb@abc.com> }
```

represents just one way of using sets and ordered pairs to express an equivalent to the English sentence.

Using the Relational Data Model could yield the table:

**p =**

| **n** | **a** | **e** |
|---|---|---|
| Alan | 42 | agb@abc.com |

The same concept could be captured by a number of XML structures:

```
<p><n>Alan</n><a>42</a><e>agb@abc.com</e></p>
<p n=”Alan” a=”42” e=”agb@abc.com”/>
<p><n>Alan<a>42</a></n><e>agb@abc.com</e></p>
```

In Extended Set Notation (XSN), which will be described in more detail later, the relationship could be written as:

$$p = \left\{ 42^{a},\ \text{Alan}^{n},\ \text{agb@abc.com}^{e} \right\}$$

Though all the above formulations of the original English expression are notationally unique, they all express exactly the same relationship between the Name (n) of a Person (p), the age of the same Person (p), and the E-mail address of the same Person (p). (It is assumed, as with any formal notation, that no symbol has more than one meaning).

In short, all these different expressions are *Informationally Equivalent.* Although this is obvious to a human, how can it be formally proven? As will be shown later, this can be done using the mathematics of Extended Set Theory.

## ***THE CONCEPT OF SCOPES***

What distinguishes extended set theory (XST) from all other set theories, and hence all mathematics derived from these set theories, is the concept of "scopes”. Intuitively, we can think of scopes as "labels” attached to each element. That is, XST elements carry with them additional information beyond whether it is in a set or not; as we shall see below XST defines an extended membership condition that means that elements are in a set *in the context of the scope*.

The power of XST comes from the ability to manipulate the "labels" along with the usual manipulation of elements, as with classical set theories. It has become apparent that these “labels” can carry information about the sequence of elements in an n-tuple (a challenge that RDBMS theorists have mostly ignored) and about the sequence and hierarchy of elements in an XML structure.

Let’s examine a Relational Data Model (RDM) example to show how XSP technology lets us *formally* handle problems that have previously been solved by *convention*. Consider the following two tables T1 and T2. While they are obviously “different” to an observer, they have exactly the same domain names and row values, and are therefore considered, under the conventions of the RDM, to represent the same “relation”. The intelligent human can easily understand the extra-mathematical convention (which , to my knowledge, cannot be defined in the mathematics underlying the RDM) that the ordering of the rows and columns is insignficant. Capturing the same notions at a formal level is considerably more challenging.

T1 =

| A | B | C |
|---|---|---|
| a | b | c |
| x | y | z |

T2 =

| C | B | A |
|---|---|---|
| z | y | x |
| c | b | a |

The challenge is in knowing what has to be preserved, what does not have to be preserved, and how to represent it in a way that is self-contained and unambiguous. Here is an XST representation, using scopes to capture the mathematical identity. (The Extended Set Notation is presented here without defining it first. This is intended only as introduction to the `look and feel' of the notation; the details will be presented later in this paper).

$$\mathtt{T1} = \{\{a^{<1,A>}, b^{<2,B>}, c^{<3,C>}\}^{<1>}, \{x^{<1,A>}, y^{<2,B>}, z^{<3,C>}\}^{<2>}\}$$
$$\mathtt{T2} = \{\{z^{<1,C>}, y^{<2,B>}, x^{<3,A>}\}^{<1>}, \{c^{<1,C>}, b^{<2,B>}, a^{<3,A>}\}^{<2>}\}$$

Note that each element in each table is labeled with information that describes its position in the structure. If we are not interested in the table structure but only in the relational content, we can label the elements in a manner that preserves only the domain in which the value occurs:

$$\mathtt{R} = \{\{a^{A}, b^{B}, c^{C}\}, \{x^{A}, y^{B}, z^{C}\}\}$$

It should be easy to see that there is a mapping from either T1 or T2 to R … simply by removing the "scope" or "label" information that is not of interest.

Let's recapitulate what has been shown here, because it is deceptively simple: In order to apply rigorous mathematics to software problems, we must start with an *informal* or conceptual model, capture its essence as a *formal* mathematical operand, and then we can *manipulate* it with rigorous operations to answer non-obvious questions. For example, the mapping of both T1 and T2 to R shows that the two tables are "informationally equivalent" according to the tenets of the relational data model.

To illustrate the issues here in more detail, consider the difficulty in representing a simple data record `[a,b,c]` as an "n-tuple", that is, the mathematical formalism usually used to model ordinary records in an application or database. The problem is in trying to formally model records with set theory (or some discipline derived from set theory, such as the relational algebra. In this case, we run into something that might be called the "fragile tuple problem."

All set theories depend on the ability to determine if a "membership condition" is **TRUE** or **FALSE**.

For an intuitive view of records, this is quite simple:

1) is "a" a member of the record [ a b c ]? Answer: yes
2) is "b" a member of the record [ a b c ]? Answer: yes.
3) is "c" a member of the record [ a b c ]? Answer: yes.

If records are to be *formally* modeled by n-tuples, where [ a b c ] is equated with <a,b,c>:

4) is "a" a member of the n-tuple <a,b,c>? Answer: ?.
5) is "b" a member of the n-tuple <a,b,c>? Answer: ?.
6) is "c" a member of the n-tuple <a,b,c>? Answer: ?.

In short, in the context of set theory "n-tuples" behave very badly simply because it is not always clear how to determine if an element "is" or "is not" a member of any given n-tuple. As long as the issue of membership does not arise, n-tuples are well-behaved, since no behavior is required of them. For more complex “records” such as XML structures that must *formally* represent both the sequence of the elements and the hierarchy of sub-elements, this problem is exacerbated.

Nevertheless, we can use a formal operation described by XSP called a “scope transform” (explained below) to solve the problem. We’ll define the n-tuple `<a,b,c>` to be the extended set $\{a^1, b^2, c^3\}$. Now we *can* say:

7) is "a" a member of the extended set $\{a^1, b^2, c^3\}$? Answer: yes, the first element.
8) is "b" a member of extended set $\{a^1, b^2, c^3\}$? Answer: yes, the second element.
9) is "c" a member of the extended set $\{a^1, b^2, c^3\}$? Answer: yes, the third element.

We can also formally define operations that re-order the “records” in <a,b,c> to <c,b,a> (which is not possible to unambiguously define with classical set theory):

$$\langle a,b,c\rangle^{/\langle 3,2,1\rangle/} = \langle c,b,a\rangle$$

Of course, there has not been a problem *doing* this transformation; programmers and SQL users have been doing it for years, and XSLT stylesheet writers can easily do this on an XML representation of the record. There has been a problem in *formally modeling* it; by means of the scope transform operation, now stupid things can do it as well as intelligent humans.

## *XML STRUCTURES AS XSETS*

XML structures can be captured as extended sets because scopes can be used to preserve the hierarchical element and attribute structure just as they are used to preserve the flat table structure. Consider the XML instance:

```
<P>
      <p>
            <n>Alan</n>
            <a>42</a>
            <e>agb@abc.com</e>
      </p>
      <p>
            <n>Mary</n>
            <a>29</a>
            <e>mky@abc.com</e>
      </p>
</P>
```

Here is an XSN representation of the same information.

$$\mathbf{P} = \Big\{\{\{\text{Alan}^1\}^{\langle 1,n\rangle},\{42^1\}^{\langle 2,a\rangle}, \{\text{agb@abc.com}^1\}^{\langle 3,e\rangle}\}^{\langle 1,p\rangle},$$
$$\{\{\text{Mary}^1\}^{\langle 1,n\rangle},\{29^1\}^{\langle 2,a\rangle}, \{\text{mky@abc.com}^1\}^{\langle 3,e\rangle}\}^{>}\}^{\langle 2,p\rangle}\Big\}$$

For a more complex example, consider the following XML structure:

```
<P>
      <p>
            <n>
            Alan
            <a>42</a>
            </n>
            <e>agb@abc.com</e>
      </p>
      <p>
            <n>
            Mary<a>29<e>mky@abc.com</e></a>
            </n>
      </p>
</P>
```

Now examine the corresponding XSN representation, using scopes to label the different nested structures:

$$\mathbf{P} = \Big\{\{\{\mathtt{Alan}^1;\{\mathtt{42}^1\}^{<2;a>}\}^{<1;n>},\ \{\mathtt{agb@abc.com}^1\}^{<2;e>}\}^{<1;p>};\\ \{\{\mathtt{Mary}^1;\{\mathtt{29}^1\};\{\mathtt{mky@abc.com}^1\}^{<2;e>}\}^{<2;a>}\}^{<1;n>}\}^{<2;p>}\Big\}$$

The distinction between attributes and elements can also be captured in XSN:

```
<P>
      <p>
            <n a = “42” e= “agb@abc.com” >
            Alan
            </n>

      </p>
      <p>
            <n a = “29” e = “mky@abc.com”>
            Mary
            </n>
      </p>
</P>
```

The  corresponding XSN representation, using scopes to represent attributes:

$$\mathbf{P} = \Big\{\ \{\{Alan^1\}^{<1;n;\{\{42^1\}^{<1;a>};\ \{agb@abc.com^1\}^{<2;e>}\}>}\ \}^{<1;p>};\\ \{\{Mary^1\}^{<1;n;\{\{29^1\}^{<1;a>};\ \{mky@abc.com^1\}^{<2;e>}\}>}\ \}^{<2;p>}\Big\}.$$

Likewise, more complex structures, with deeply nested elements and information represented as a combination of attributes and elements, are just a matter of degree and do not depend on any new structural requirements.

```
<P>
  <p>
    <n>Alan<z>Z</z>G.<w>W<y d='H'>Q</y></w>Brown</n>
    <a>42</a>
    <e>agb@abc.com</e>
  </p>
  <p a="29" n="Mary" e="mky@abc.com" />
</P>
```

$$\mathbf{P} = \Big\{ \{\{Alan^1;\ \{Z^1\}^{<2;z>};\ G.^3;\ \{W^1;\ \{Q^1\}^{<2;y;\{\{H^1\}^{<1;d>}\}>}\}^{<4;w>};\ Brown^5\}^{<1;n>};$$
$$\{42^1\}^{<2;a>};\ \{agb@abc.com^1\}^{<3;e>}\}^{<1;p>};$$
$$\emptyset^{<2;p\{Mary^1\}^{<2;n>};\ \{29^1\}^{<1;a>};\ \{mky@abc.com^1\}^{<3;e>}>}\Big\}.$$

Thus, we see that XSP Technology provides a formal system and notation that contains a means, called scopes, to label the elements in an extended set so that structural information can be preserved (or not) as needed. Furthermore, this formal system is rich enough to handle both XML and relational data in the same notation system. One might argue that transforming a non-mathematical notation such as XML syntax into an arcane mathematical notion may look impressive, but achieves little more than destroying the familiar syntax. It is true that mathematical notation in and of itself does not guarantee the soundness of the represented concepts. The hostile mathematical notation must provide concrete benefits in order to justify the translation.

The benefit of capturing the mathematical identity of XML structures in terms of Extended Sets (Xsets) is that all Xsets are mathematically sound operands that behave predictably under Extended Set Operations, Xsops. Since Xops are just extensions of set-theoretic operations that work on sets of "labeled elements", all the mathematical power of set theory can now be harnessed to design, develop, implement, and use systems intended to process XML structures.

Further details on the representation of XML in extended sets can be found in [1].

## *INFORMATION EQUIVALENCE AND SCOPE TRANSFORMS*

In essence, XSP technology is a mathematical system that redefines sets and set operations in terms of scopes, and a methodology for using scope transforms to formally handle a wide variety of previously intractable mathematical modeling problems.

The concept of Information Equivalence is of great practical importance, but difficult to address without using scopes. Consider, for example, the sentence "Mary is 29 years old and her e-mail address is mky@abc.com". No mention is made whether an XML structure or an RDM relation captures this relationship, but let's assume one of the two. Now let's ask the question, "Is Mary older than Alan?" All the "relationship" information is available to answer the question, but how can we go about doing so?

This question could be answered either "formally" or "constructively". Most readers can easily envision a constructive answer, e.g. a SQL query on an RDM representation of the data or an Xpath query on an XML representation of the data. Unfortunately, the machinery necessary to answer the question formally is not in place, or at least not readily available. This paper will present a mechanism called a "Scope Transform" based on the XSN representation of the data, and implemented in the XSP Technology.

In the case of “Alan” and “Mary”, let People (P) be the set of Persons (p). We have seen how in XSN, ‘a’, ‘n’, ‘e’, and ‘p’ are called “Scopes”. We can posit the existence of some function “f” that extracts the information needed to answer our question, that is:

$$\mathbf{F}(a,n,p,Mary,\mathbf{P}) \;<\; \mathbf{F}(a,n,p,Alan,\mathbf{P}) = \mathbf{TRUE}$$

In the formal environment of XSP Technology, we can *actually define* this function in terms of operations on the extended sets, as we shall see later.

$$\rho_a(\rho_p(\mathbf{P}|\{\{Mary^n\}^p\})) < \rho_a(\rho_p(\mathbf{P}|\{\{Alan^n\}^p\})) = \mathbf{TRUE}$$

This function will work with *any* of the structural definitions of “Alan” because it consists of mathematical operations on mathematical operands, independently of the structure of the data. That is, any of the structures can be mapped to the XSN representation of the record; they are Informationally Equivalent, so the results of the operations are equivalent. In other words, this operation is a well-defined *algebraic* transformation on mathematical entities rather than an *algorithmic* recipe for manipulating data structures.

The “essence” of XSP development is in defining appropriate XST representations of data and modeling the operations on those extended sets as *Scope Transforms*. “Transform” means “to change in composition and/or structure.” Of course, the notion of changing the structure of a set is counter-intuitive to those familiar with Classical Set Theory. In CST, it is not possible to “change a set” any more than it is possible to “change an integer”. One can replace one set by another, just as one can replace one integer by another, but adding 1 to 6 does not change 6 to 7. Adding or deleting an element from a set just generates a different set. So, “transforming” a set means to “replace” the set with another.

Thus, XSP provides the foundation for an *Integration Technology* that can underlie all applicable formal technologies relevant to information processing: XML and the Relational Data Model, and also graph theory, lambda calculus, category theory, denotational semantics, fuzzy set theory, and others. That is, the operands of all these disciplines can be represented as extended sets, and their operators modeled as scope transforms. In this way, the practical realities of real software – as seen by the user and as physically implemented in machines – can be represented in a formal system that is rich enough to “absorb the complexity” of XML and RDBMS into a common framework.

## Meeting the Challenges of XML with XSP Technology

XML has generated an enormous amount of enthusiasm in the database and middleware industries in the last few years. XML *structures* (a generic term used here to denote both “documents” and “data”) are widely used to define, store, and exchange data.

Nevertheless, XML faces significant challenges that must be overcome as it matures. These include:

- Modeling
- RDBMS Integration
- Distributed processing
- Data/Metadata integration
- Data Independence
- Interoperability

*Formal Modeling* – There is no well-accepted way to map an application’s view of its data onto an XML representation in a database. Various issues that are covered in every textbook treatment of RDBMS data modeling – such as how to minimize redundancy and maximize referential integrity – have no well-defined answer in XML. In fact, we’re starting to see respected data modeling theorists point out this unpleasant fact to the XML community. [2]

*RDBMS - XML integration* -- We now have a considerable "impedance mismatch" between RDBMS and XML systems. The typical approach is to produce an XML view of RDBMS data in order to integrate it with XML data. This works, inefficiently of course, and loses the formalisms and methodologies of the relational world in the process. This is a significant theoretical issue now, and will become even more important as XML technology makes more and more inroads into places where RDBMS concepts are taken for granted. We need a unified formal model -- and perhaps some formal methodologies -- to support both XML and RDBMS systems design and processing.

*Data - Metadata integration* -- New models and formats (such as RDF and XML Topic Maps) for incorporating metadata to describe the "semantics" of XML documents and data can in principle help make it easier to perform intelligent queries. Unfortunately, XML is mainly used as a serialization format for these; they each have their own information model, which in the short run complicates rather than simplifies the task of performing intelligent queries. We need to be able to describe the RDF and XTM paradigms as well as XML structured documents and RDBMS databases into a unified formal model of information.

*Distributed Processing* – The development of the Internet, in particular, has led to demands for de-centralized and “peer to peer” distribution of data and processing resources. Ideally, we need a processing architecture that broadcasts operations to distributed processors and retrieves useable results in response, rather than requesting data from remote servers. Neither the relational model or the XML data model supports these demands easily, but this limitation will become more and more important as we go forward.

*Data Independence* – Much of E.F. Codd’s work stresses the importance of “data independence”, i.e. the principle that programs should continue to run, and the user’s view of data be unaffected, as the logical and physical structure of databases evolves. This is better supported in today’s relational databases than in XML databases – SQL queries will not be affected if the columns of a table are re-ordered, or if additional columns are added to a table, or a table is re-partitioned across physical storage devices; XML operations are quite likely to fail if attributes are re-defined as elements, or elements added or re-ordered.

*Interoperability* – The XML world faces increasing challenges because the XML meta-model is used to describe structurally different, but semantically equivalent documents and messages. For example, there are numerous different “standards” proposed to describe the XML structure of various business documents – invoices, purchase orders, catalogs, etc. It is unrealistic to expect that any one standard will prevail in most industries; and few real businesses confine their operations to a single industry anyway. For example, an electronic component manufacturer may deal with suppliers that sell directly to a variety of manufacturers, not just those within the “electronics industry”, may produce components sold to companies producing very different products (e.g., autos, airplanes), and will have to borrow money and make payments via the “financial industry.” Even if the electronics, automobile, aircraft, and financial industries all come up with well-established XML message and document formats, this company will have to interoperate with some complex mixture of them.

In short, the biggest challenge facing XML vendors and users today is to *integrate* XML processing with legacy RDBMS systems, across different XML vocabularies that represent the same “real” concepts, with semantic metadata (RDF, Topic Maps, etc.), across distributed heterogeneous platforms, and across the conceptual divide between VIEW as seen by an intelligent human and the INTERNAL processing performed by a dumb machine. This requires us to define a formal environment that can capture the essence of all that needs to be integrated, i.e., it must integrate the various layers of formal foundations that are already being used.

## *Getting XML Into a Formal Framework*

Many think of XML as “well-defined” model of data, with strict rules for “well-formedness”. It is true that by the standards of the computing industry, XML is fairly clean and reasonably well-defined, and as a practical matter it has been “good enough” to launch a significant sub-industry. It is also true that the mathematics of graph theory (and related disciplines, such as “hedge automata”) provides a useful conceptual model and suite of algorithms that have been applied to XML. Nevertheless, these pragmatically useful *algorithms* are not built on a formal algebra, that is, a mathematically well-defined system of operations and operands. To put it in the terms used here, graph theory provides a formal mechanism for defining structures, but the only operations defined on those structures are those to navigate them (getNext, getParent, etc.). We need to be able to capture the mathematical identity of an XML structure in a way that lets us perform meaningful operations on the information it contains as well as the structures representing the information.

## *Structure-Centric and Operation-Centric Processing*

We shall argue that these challenges can be addressed by redirecting attention away from the *structure* of XML (and RDBMS) data and on to *operations on a mathematical representation* of XML data and documents The “Structure-Centric” information-processing approach focuses on *manipulation of data structures.* Loading information from storage implies wrapping structures around data; finding information implies navigating the data structures; changing data implies rearranging the data structures. Optimization generally builds new data structures (such as indexes); distributed processing implies transferring data around a network. Structure-centric systems tend to be fragile because every change to data formats or data structures must be propagated throughout the system.

The “Operation-Centric” approach focuses on *operations on operands and* is characterized by:

- A focus on a rich set of operations that work on mathematically-defined objects
- Strong Data Independence that provides a common mathematical foundation for modeling data as viewed by an application programmer, as processed by a computer, and as it is physically stored.

Let’s examine why set processing can serve as the formal basis for an operation-centric architecture: Any concept or structure that ultimately gets implemented on a digital computer is actually embedded in a physical environment that is intrinsically a set-processing environment. At the very lowest conceptual level, all digital data is a "bit string", or set of bits. All operations performed by a computer can be described as one set of bits being operated on to produce another set of bits. Thus, all operations in any computer program are ultimately are set operations, and all observable computer behavior can be described as set processing. .  Set operations are known to be functionally 'rich' and logically consistent. This means that if terms commonly used in software development such as 'information', 'relations', 'objects', 'lists', 'stacks', etc. could be defined as sets, then any set operation defined on or between them could be "bug free", and all known set operations would apply to them!

The relational data model (RDM) also exhibits some of these characteristics: Codd’s model provides operation-centric *convenience* to the application programmer via an algebra that works on “tables” and hides the messy underlying reality of the execution and storage environments.  This operation-centric view has provided enormous benefit to several generations of programmers, and the approach taken here should be seen as an attempt to complement Codd’s work, not to replace it. Nevertheless, as we know, the relational model is not well suited for modeling hierarchically nested data such as XML.  Likewise, the relational model does not provide an operation-centric *architecture* because the RDM operations only apply to tables at the application level, and not to the physical data at the processor or storage level.

The Object-Oriented (OO) technologies also exhibit some of these characteristics.  Well-designed objects should provide clean abstractions that hide (“encapsulate”) their underlying data structures.  Likewise, the methods in an OO system provide the operators, and the objects provide the operands, so a “pure” OO system can be seen as “operation centric”.  Again, few real OO systems exhibit Strong Data Independence because the implementation of the abstractions and methods is left to the skill of the programmer rather than being based on a hierarchy of mathematical definitions.

The world of XML-based data processing certainly incorporates some insights and techniques from RDM and OO technologies – XML data is often “normalized” for RDBMS storage, and OO principles have been applied to a certain extent in the XML “Document Object Model” API standard.  This API, however, merely provides operations to navigate and manipulate a specific XML *tree structure*, so in effect this is a structure-centric technology; DOM applications typically show little data independence and easily “break” as the underlying XML structures change.

There are also XML “data binding” tools that allow Java or C++ object definitions and load/save methods to be automatically generated from XML schema.  Such tools essentially relegate XML to a data interchange format, however, and do not even attempt to provide an operation-centric approach to the handling of a generic XML data model.

To summarize: developments in information technology over the past 25 years or so, especially the RDM and OO technologies, have indeed moved us in an “operation centric” direction.  XML has been widely accepted because of its universality, interoperability, and ability to directly represent documents and data without the normalization process required by the relational model, but has to a certain extent moved us back to the structure-centric era when it comes to formal models and methodologies. We are faced with a challenge to achieve “strong data independence” while maintaining the powerful and extremely useful XML data model, and this report attempts to help meet that challenge.

This has utterly practical implications for XML users and vendors who may not care about abstract notions such as “operation centric processing ” in their own right:  There are good reasons (borne out by close to thirty years of experience with XSP Technology) to believe that the operation-centric approach can produce applications that perform better, scale better, and can be maintained and distributed more easily than those that rely on a shared understanding of structure.  This will become increasingly important as XML becomes more widely used, because it pushes the boundary of structure-centric approaches, and doesn’t fit neatly into the Relational Data Model paradigm that has, to some extent, provided an operation centric approach to mainstream software developers for more than 20 years. To put it most bluntly, software works better when it is design to perform operations rather than chase structures, and it is more efficient to broadcast operations across a distributed application than to exchange structures.

To illustrate this point, consider the well-known technique of building indexes to speed up database searches. Indexing is almost universally considered to be a "solution" not a "problem". There are may situations where indexing is the ideal tool for the job. There are many more situations where indexing is the wrong tool for the job. If there are no other tools to choose from, then indexing has to be used by default.

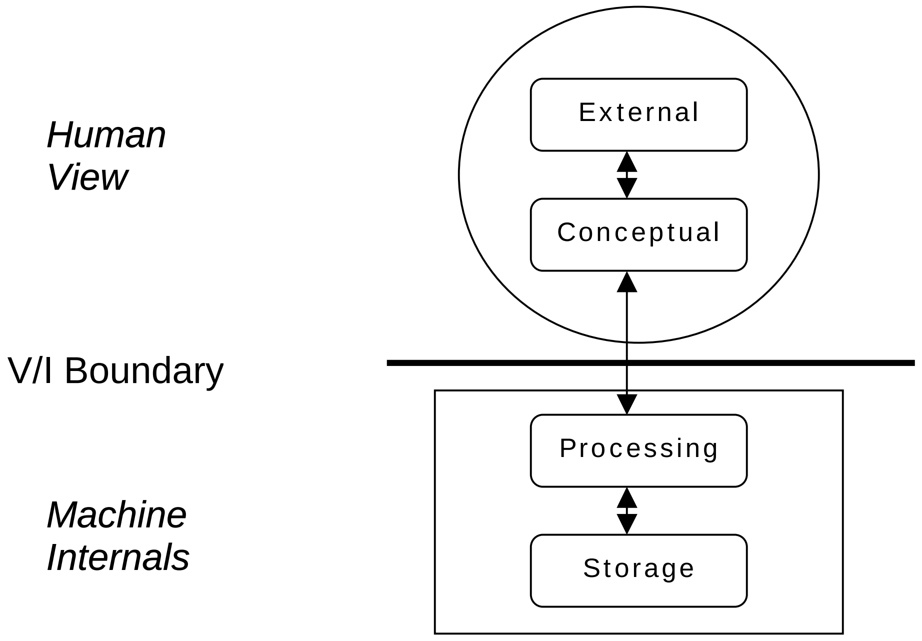
The default case is so prevalent today, that even seasoned database experts will argue that "indexing" is THE way to obtain high-performance data access. It is as difficult to convince a database expert that "indexing" can be inefficient as it is to convince a mathematician that "n-tuples" are not well-defined.

The argument to refute this is not very difficult to understand, but it has to be explained in terms of "operation-centric" systems, which are orthogonal in nature to the "structure-centric" systems that databases have been dependent on for the past 30 years. A detailed example involving one of the TPCD benchmark datasets and queries is presented below, but in a nutshell the argument goes like this: As a series of well-defined operations on well-defined operands proceeds, the result sets of the operands gets smaller. That is, if we have an initial set containing a large number of operands, and progressively extract subsets meeting some formal criteria, the set of operands gets smaller and smaller, so the operations get faster and faster. But if we are operating on physical data structures (even inverted indexes, hash tables, or whatever) they don’t get any smaller on every pass through them, so there is no speedup effect.

## ***Strong Data Independence***

The “classical” definitions of data independence go hand in hand with the ANSI/SPARC conceptual model of a database system. (*ANSI/X3/SPARC Study Group on Data Base Management Systems* ACM Special Interest Group on Management of Data, 1975.). The ANSI/SPARC framework defines three levels at which one can examine a database system: External (the database as seen by a user or GUI application), Conceptual (the database view common to all applications, including business rules, mappings, etc.), and Internal (the physical representation of the database in the computer and on disk). The boundary between each level can be thought of as an “abstraction firewall” that should not be violated if data independence is to be maintained. Data independence in the ANSI/SPARC data model is defined at the External to Conceptual or E/C boundary, and implies that applications should not require changes if the “conceptual schema” changes, e.g., as entities not in an external view are added or deleted. Strong data independence is desired at the V/I boundary, and implies that the human view of the data should not have to change if the internal representation changes, e.g. files are re-partitioned across physical disk devices.

*Strong data independence* refers to the boundaries in a slightly different manner: It implies that there is *a precise mathematical interface* across the V/I boundary, sometimes called the Internal Level Interface. [3]



The higher levels of the ANSI/SPARC framework provide useful abstractions of the lower levels, and the lower levels provide concrete details that are necessary for systems administrators and DBMS implementers to understand but can be hidden from users. The "classic" ANSI/SPARC framework says nothing about *how* the abstractions at one level are mapped onto the details at another, whereas the notion of strong data independence requires the mappings from a higher to lower level of abstraction to be *mathematically sound operations*. Similarly, the relational algebra specifies operations on tables that (when Codd first wrote) had no obvious relationship to any actual data structures in a hierarchical or CODASYL DBMS. The mapping was originally "an exercise left to the reader", and once RDBMS systems came into existence, the mapping became a job for programmers at Informix, Oracle, etc. to define in proprietary code. The principle of strong data independence, by contrast, insists that these mappings be formal and explicit.

Thus, the ANSI/SPARC and idea of strong data independence are completely compatible, but reflect different priorities: ANSI/SPARC is more a conceptual model for application programmers trying to hide the underlying complexity, but we want to evolve it into design tool for database implementers working to manage the various sources of complexity.

To achieve strong data independence between any two distinct environments requires a technology to provide a common mathematical representation of both the data to be operated on and the operations to be performed on the data. What gives XSP the ability to model operations and operations on both sides of the V/I boundary, that is, in both in VIEW and INTERNAL environment?

The answer is *scopes*. As we saw in the first section, adding labels to the elements of classical set theory allows sequences and hierarchies to be formally represented and manipulated. This in turn allows us to add more mathematical rigor to the theory underlying relational databases, to represent hierarchical objects such as XML structures in a way that allows XML processing to be operation-centric, and provides a language that can be mapped all the way down to the physical memory, processor, and disk of a computer.

# XSP Technology Overview

A brief exposure to use of extended set notation (XSN) should provide some intuitively accessible exposure to the extended formal modeling capabilities available through the use of "labeled elements".

## *CLASSICAL SET THEORY*

Let's begin with a review of the concepts and terminology of set theory as we learned it in school, which we shall refer to here as *Classical* Set Theory or CST.

Any item that can be defined solely in terms of its membership can be mathematically identified as a *set*. Given any collection of sets, then any operation involving the sets that can be defined solely in terms of the membership of the respective sets is a *set operation*. *Set processing* is defined to be the manipulation of sets by set operations. Thus, set processing depends on one and only one concept, membership.

In axiomatic set theory, *membership* is a 'given' that is used to define all other concepts. That is, "membership" itself is never defined, and any interpretation that is consistent with the axioms is a legitimate interpretation of the term. Informally, membership is equivalent to "belonging to", and the terms "member of a set" and "element in a set" are synonymous. The practical utility of the term membership is that it allows any item for which membership is defined to have mathematically sound operations defined on it. In computer parlance, "mathematically sound" means "bug free". This does not, of course, mean that every implementation of a set processor is bug free, only that every implementation that preserves the membership behavior of the operation will be free from logical inconsistencies. This cannot be said for operations that are not known to be mathematically sound.

Let's review the usual notation and conventions used to describe sets and set operations.

A set Q with the elements x, y, and z is denoted Q = {x, y, z}.

A set without any elements is called the "null set", and denoted with the symbol $\varnothing$.

For any set, Q and any item, x, the assertion "x is an element of Q" is always either **TRUE** or **FALSE**.

The property "to be an element of a set" is denoted as the predicate symbol $\in$Thus, $x \in Q$ means "x is an element of Q", and $w \notin Q$ means "w is not an element of Q".

According to the *extensionality principle,* two sets Q and R are equal, if and only if they have the same elements; the order and repetition of the elements doesn't matter. Thus {A, B, C} = {B, C, A} = {A, A, A, B, B, B, C, C, C}

We can, however, talk about ordering, such as an *Ordered Pair* (or simply "*pair*") denoted <a,b> where a is the first element and b is the second.

The *cardinality* of a set is the number of unique elements in the set. Thus {A} has a cardinality of 1, {A,B} has a cardinality of 2, and {A,A,B} also has a cardinality of 2.

A set Q is a *subset* of set R if every element in Q is also an element of R. This is denoted $Q \subseteq R$. Thus, there is a special case where if Q = R so $Q \subseteq R$. A set Q is a *proper subset* of R if $Q \neq R$. This is denoted $Q \subset R$.

All operations, defined strictly in terms of a set's members, are set operations. Every set operation is defined for every set, and the result is a set. The most basic and useful set operations include:

*Intersection*: The intersection $Q \cap R$ of sets Q and R results in a set consisting of the elements that are members of *both* Q and R.

*Union*: The union $Q \cup R$ of sets Q and R results in a set consisting of the elements that are members of *either* Q or R.

*Difference*: The difference Q - R of sets Q and R results in a set consisting of the elements that are members of Q but not members of R.

*Symmetric Difference*: The symmetric difference $Q \Delta R$ of sets Q and R results in a set containing the elements that are in Q or R but not both.

*Cartesian Product*: The Cartesian product Q x R of sets Q and R results in a set containing all pairs with the first element in Q and the second element in R.

## *THE EXTENDED MEMBERSHIP CONDITION*

Classical set theory deals with a membership concept in which something is either "there" or "not there". Extended set theory adds an additional condition to the membership concept in which something is either "there under the specific condition" or "not there under the specific condition".

Symbolically, the distinction between CST and XST can be made very simply. Let E(A,x) express the assertion "x exists in A". Then E(A,x) is either true or false depending on whether "x" really exists in A or not.

Now in XST we can add the condition y such that the expression E(A,x,y) is either true or false depending on whether "x exists in A under condition y" or "x does not exist in A under condition y". Thus, E(A,x) reflects CST membership while E(A,x,y) reflects XST membership. By convention we can subsume CST under XST by assuming "no additional condition" as represented by zero, "0". Thus all CST membership conditions E(A,x) can represented in XST by E(A,x,0).

As we will see shortly, it is this additional membership dimension that allows XST to model structures and operations that actually exist in a computer, but that cannot be modeled CST. We use 'superscript' notation to capture the extended membership condition or *scope* along with the *element* as the condition of membership.

The notation:

$$A = \{x^a,\ y^b,\ z^c\}$$

is equivalent to the expression "the extended set A is composed of the elements x under constraint a, y under constraint b, and z under constraint c".

Likewise:

$$x \in_y A$$

is equivalent to the expression "x is a member of set A under constraint y"

Thus, to put it somewhat simplistically, membership in Classical Set Notation has a dependency on just one condition – *element*. Extended Set Notation has a dependency on two conditions – *element* and *scope*. CSN can be subsumed under XSN since every binary truth functional for a CSN set definition $\Gamma(x)$ can be re-expressed as a special case of the ternary truth functional for an XSN set definition $\Gamma(x, y)$ where y is null.

"*Scope*", like "*element*" is an undefined term, and XSN is purely a syntactical framework for defining consistency of operations on well-defined objects. Any "meaningful" situation modeled by XSN can enjoy the results of this operational consistency, but not because of any "meaningfulness" of the definitions of "element" or "scope". In the work discussed below, *scope* generally refers to a specific set that can be given a "meaning" in terms of the information contained in an XML document, but it must be stressed that this is not inherent in the definitions of XSN. Other interpretations are possible; for example, one can subsume "fuzzy set theory" under XST by defining the scope condition as "degree of membership."

## ***SCOPE OPERATIONS***

All CST set operations are extended set operations (with a null scope for all elements), and there are XST analogues for the basic set operations (intersection, union, difference, symmetric difference, etc.) that include both parts of the extended membership condition. There are also a number of operations that are unique to XST

*Scope Restriction*: The scope restriction $X^{[\sigma]}$ is equal to the set of scoped elements in X where the scopes are restricted to those in the set $\sigma$. That is,

$$\mathbf{X}^{[\sigma]} = \{y^s : y \in_s x \ \& \ s \in \sigma\}$$

For example:

$$X = \{ a^1, b^2, c^3, d^4\}$$
$$\sigma = \{ 2,3,6\}$$
$$x^{[\sigma]} = \{b^2, c^3\}$$

*Set Scope Restriction*: The set scope restriction $X^{[\sigma]*}$ the scope restriction applied "one level down." That is,

$$\mathbf{A}^{[\sigma]*} = \{x^s : (\exists z)(z \in_s A) \ \& \ (x = z^{[\sigma]})\}$$

For example:

$$A = \{\{a^1, b^2, c^3, d^4\}\}$$
$$\sigma = \{2,3,6\}$$
$$A^{[\sigma]*} = \{\{b^2, c^3\}\}$$

*Scope Transform*: The scope transform $X^{/\sigma/}$ is equal to the set of scoped elements in X where the scopes are restricted to those in $\sigma$ transformed by the scope in set $\sigma$. That is,

$$\mathbf{X}^{/\sigma/} = \{y^t : (y \in_s \mathbf{X}) \ \& \ (t \in_s \sigma)\}$$

For example:

$$\begin{aligned} X &= \{a^1, b^2, c^3, d^4\} \\ \sigma &= \{1^2, 3^4\} \\ X^{/\sigma/} &= \{b^1, d^3\} \end{aligned}$$

*Natural Restriction*:

$$\mathbf{Q}|\mathbf{A} = \left\{z^s : (\exists a)(a \in_s \mathbf{A} \;\&\; z \in_s \mathbf{Q}) \;\&\; (a \subseteq z)\right\}$$

*Element Projection*:

$$\rho_{\mathbf{s}}(\mathbf{A}) = y \Leftrightarrow y \in_s \mathbf{A} \;\&(\forall z)(z \in_s \mathbf{A} \rightarrow z=y)$$

Now we have defined enough of the XSP mathematics to show how the “is Mary younger than Alan” question can be answered: Extracting only the relationship information and ignoring the XML structure in the previous example, we can represent the relationship in the form:

$$\mathbf{P} = \left\{\left\{42^a, \text{Alan}^n, \text{agb@abc.com}^e\right\}^p, \left\{29^a, \text{Mary}^n, \text{mky@abc.com}^e\right\}^p\right\}$$

Informally, we’re looking for a function that will show that Mary’s age is less than Alan’s age:

$$\mathbf{F}(a,n,p,\text{Mary},\mathbf{P}) < \mathbf{F}(a,n,p,\text{Alan},\mathbf{P}) = \textbf{TRUE}$$

Expressed in actual XSP operations, this is:

$$\rho_a(\rho_p(\mathbf{P}|\{\{\text{Mary}^n\}^p\})) < \rho_a(\rho_p(\mathbf{P}|\{\{\text{Alan}^n\}^p\})) = \textbf{TRUE}$$

# Conclusion

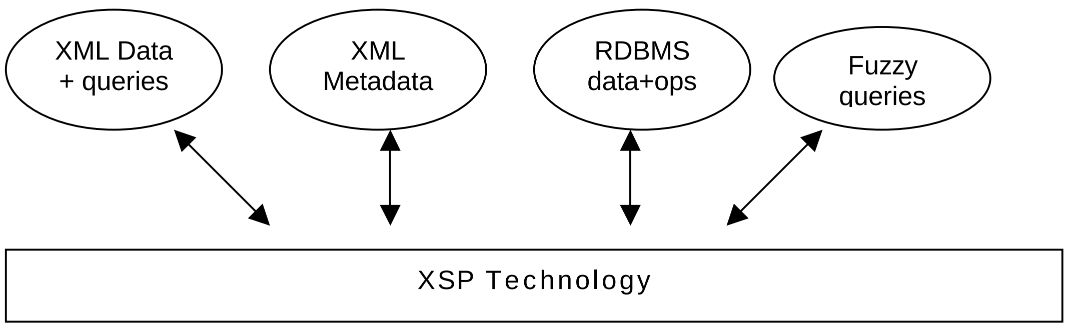
As we have seen, concept of a “scope” is at the heart of XSP Technology. The significance for XML is that scopes can be used to model ordered and nested relationships that simply cannot be handled cleanly in set theory and similar formalisms. Furthermore, scopes can be used to model the addressing mechanism of a computer, allowing the use of XSP technology to formally represent and manipulate the "real" operations of a computer in a useful way. This allows us to rigorously map the operations and operands of relational algebra, XML processing, etc. onto a model of a physical computer all the way down to the level of the secondary storage, i.e., on both sides of the “V/I Boundary” discussed above.

## *MEETING THE CHALLENGES OF XML WITH XSP*

How does XSP technology address the challenges faced by XML noted at the beginning of this section?

- Modeling - One could use XSN to formulate a set of operands (i.e., a data model) and operations that are rich enough to *formally* capture the structures of and useful manipulations of XML documents.
- RDBMS Integration – XSP technology is rich enough to subsume both XML and the relational model, so the informational equivalence of different XML schemas, or an XML schema and an RDBMS schema, can be captured.
- Distributed processing - The operation-centric, data-independent processing architecture promoted by XSP technology make it feasible to “broadcast operations” rather than “transmit structures” across systems and platforms.
- Data/Metadata integration – Just as XSP technology can subsume both XML data and RDBMS data, it can subsume XML data and various metadata formats into a common “informationally equivalent” vocabulary.
- Data Independence – XSP technology allows the traditional RDBMS concept of data independence to be extended to the XML world, so that XML users do not have to *care* (while still preserving the information in case they *want* to care!) about structural differences such as whether elements are children or siblings of one another, or whether information is represented in XML as an attribute or an element.
- Interoperability – The mathematical formalisms required to use XSP technology allow a more rigorous definition of XML-related specifications, which should allow them to be more easily implemented in a provably correct, hence interoperable manner.

More importantly, XSP is best thought of not as a “better XML than XML”, but as an integration technology that can bring together XML and other approaches to data representation, searching, and manipulation. As XML technology is more widely used, there will be increased demands to use that data in conjunction with XML-related metadata formats (RDF, XML topic maps, Xlink linkbases), relational data and SQL queries, and to employ “fuzzy” techniques to handle the inaccuracy and ambiguity inherent in real data.



XSP Technology can represent XML data, metadata, relational data, and fuzzy logic assertions together within a common framework: the main difference is in how the “scope” component is interpreted, whether it be as a reflection of a piece of information’s relationship to an XML structure, a system of metadata represented as XML, as membership in a relational domain (or perhaps its physical location in a table), or even as the piece of information’s degree of membership in some fuzzy set.

## *LOOKING AHEAD*

The work in reference [1] put more mathematical meat on the bare bones of an introduction to XSP technology outlined here. Additionally, there is a “proof of concept” implementation of XML processing using XSP technology that this really does work at the VIEW level of mathematics and at the INTERNAL level on a physical computer.

What else is needed to make XSP technology a widely useable, practically viable underpinning for XML developers? The research described above has shown that XML can be modeled and manipulated with XSP technology, but it requires significant mathematical sophistication on the part of the user to do so. The demonstrations require a VIEW of the XML data right at the level of the INTERNAL representation of the extended sets with scopes representing the physical address offsets of the data.

As noted above, mathematics is best used to “absorb complexity”, and higher-level XSP operations need to be defined (at roughly the level of abstraction of the relational algebra) that will absorb some of this complexity.

References

[1] D L Childs, “XSP Technology for XML Systems Design and Development,

[ 2] See http://xml.oreilly.com/news/xmldevcon_0201.html for an account of Peter Chen’s speech at XML DevCon 2001or Chen’s comments at a “State of XML” panel discussion available at http://technetcast.ddj.com/tnc_play_stream.html?stream_id=555.

[3] See also D L Childs “Mathematical Identity: The Essence of XSP Technology”