How to Develop Stylesheet for XML to XSL-FO Transformation

July, 2005
2nd Edition

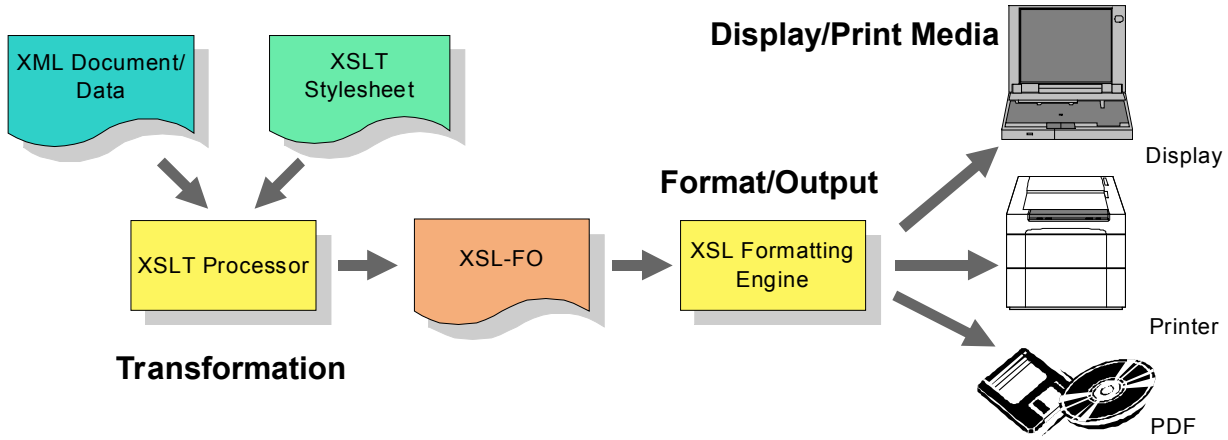
Antenna House, Inc.

Table of Contents

Preface

Extensible Stylesheet Language (XSL) (Source Information [1]) has been brought to the attention of a wide audience as a specification recommended by W3C on 15th October, 2001 for displaying and printing the XML document. The following is the general process to transform the XML document into XSL Formatting Objects (XSL-FO) and print it.

1. Develop the stylesheet that conforms to the DTD of the source XML document to create the target output.
2. Input the XML document and the XSL stylesheet to the XSLT processor to create XSL-FO.
3. Get the target outputs such as paper-based output, PDF output, by the XSL-FO processor.



Generating XSL-FO and Display/Print by XSL Formatter

The knowledge about XSLT and XSL is necessary to develop XSLT stylesheet. The knowledge about XSLT and XSL is necessary to develop XSLT stylesheet. As for XSLT, many reference books has been published other than the specification (Source Information [2]) . Probably, XSLT is already much more familiar one, since it is often used for the conversion from XML to HTML. The XSL specification (Source Information [1]) has a huge amount of contents, it has over 400 pages. It is pretty hard to understand this specification. But basically it is intended for implementers. It is not necessary for XSLT stylesheet designers to understand everything. You can fully write stylesheet by knowing some regular contents and patterns.

This report explains how to edit stylesheet which is used for transforming XML documents into XSL-FO according to the example of SimpleDoc. SimpleDoc is made for this report as a format to write a simple document. This is based on PureSmartDoc (Source Information [4]) presented by Tomoharu Asami. To make it a simple, the number of the elements is reduced and the useful functions for writing and formatting documents are added.

This report explains how to edit a stylesheet which is used for transforming a SimpleDoc document into XSL-FO. This report itself is the instance XML document of SimpleDoc.dtd, and it is then ready to be formatted by Antenna House XSL Formatter with the stylesheet which transforms the SimpleDoc document explained here into XSL-FO.

Step for XSL-FO Transformation

Now, what steps are necessary to develop XSL stylesheet? These steps are explained briefly as below.

Steps	Contents
Know the structure of the XML document	First, the information about the structure of XML source documents is required. XSLT processor can transform XML document into XSL-FO without a DTD. But the information described in the DTD such as types of elements, contents of elements, appearing order of elements and values of properties are necessary for developing a stylesheet.
Specify a printing form	This is the printing form as a final output, in other words the output specification. XSL is a formatting specification. Printing forms has various range of specifications such as sizes and layouts of printing paper, layouts of head and body, deciding whether or not to output index, table of contents, and so on.
Apply a printing form to formatting objects	After determining the specification of printing, you have to know what XSL formatting objects and properties are applied in order to print in this style. It is better to practice how to specify by referring to a simple stylesheet.
Develop an XSL stylesheet	Put the instructions to the stylesheet in order to transform XML source documents into the target printing form. Map the XML source document to XSL formatting objects that can generate the output specification. The stylesheet have the similar aspect as the general programming languages, while it may be difficult if you do not understand the feature of the XSL. (1)

(1) Refer to the definition list template in this report. In XSL, Structure for control of the conditional branch can be made, but it is impossible to assign a value to a variable. The technique to realize by calling loops recursively is necessary.

SimpleDoc Organization

The following table shows the structure of SimpleDoc treated in this report. For more detail, refer to SimpleDoc.dtd.

Element	Meaning	Definition
a group of block elements	−	p | figure | ul | ol | dl | table | program | div
a group of inline elements	−	a | note | span | b | i | em | code | br
doc	root element	(head, body)
head	header	(date | author | abstract | title)*
date, author, abstract, title	header elements, date, author, abstract, title	(#PCDATA | a group of inline elements)*
body	body	(chapter | part | section | a group of block elements | a group of inline elements)*
part	part	(title, (chapter |a group of block elements | a group of inline elements)*)
chapter	chapter	(title, (section | a group of bock elements | a group of inline elements)*)
section	section	(title, (subsection | a group of block elements | a group of inline elements)*)
subsection	subsection	(title, (subsubsection | a group of block elements | a group of inline elements)*)
subsubsection	subsubsection	(title, (a group of block elements | a group of inline elements)*)
title	title	(#PCDATA | a group of inline elements)*
p	paragraph	(#PCDATA | a group of block elements | a group of inline elements)*
figure	figure	(title?) Specify a file by the src property.
ul	unordered list	(li*) Specify a character for label of line by the type property.
ol	ordered list	(li*) Specify format of number in the label by the type property.
dl	definition list	(dt, dd)* Specify whether to format the block in horizontal way or in vertical way by the type property.
dt	definition term	(#PCDATA | a group of block elements | a group of inline elements)*
dd	description of details	(#PCDATA | a group of block elements | a group of inline elements)*
table	entire table	(title?, col*, thead?, tfoot?, tbody) Specify whether to make auto layout or fixed by the layout property. Specify the width of the entire table by the width property.
col	column format	EMPTY Speciry the number of the column by the number property, the width of the column by the width property.
thead	table header	(tr*)
tfoot	table footer	(tr*)
tbody	table body	(tr*)
tr	table row	(th | td)* Specify the height of the row by the height property.
th	table header	(#PCDATA | a group of inline elements | a group of block elements)*

Element	Meaning	Definition
		Specify the number of the columns to be expanded across, the number of the rows to be expanded vertically by the colspan and rowspan properties. The align property allows horizontal alignment to be set to left, right, or center. The valign property allows vertical alignment to be set to top, middle, bottom or baseline.
td	table data	(#PCDATA | a group of inline elements | a group of block elements)* Specify the number of the columns to be expanded across, the number of the rows to be expanded vertically by the colspan and rowspan properties. The align property allows horizontal alignment to be set to left, right, or center. The valign property allows vertical alignment to be set to top, middle, bottom or baseline.
program	program code	(#PCDATA | title)*
div	general block element	(title, (general block element | general inline element)*) The div element expands the type by the class property.
a	anchor(link)	(#PCDATA | a group of inline elements)* Specify URL as the value of href property.
note	note	(#PCDATA | a group of inline elements)*
b	bold typeface	(#PCDATA | a group of inline elements)*
i	italic typeface	(#PCDATA | a group of inline elements)*
em	emphasis	(#PCDATA | a group of inline elements)*
code	program code of the in-line elements	(#PCDATA | a group of inline elements)*
span	general in-line element	(#PCDATA | a group of inline elements)*
br	line break	EMPTY

The features of SimpleDoc are:

- You can start writing a document from part, also start from section. It has a flexible structure so that it can map to various kinds of documents.
- The number of the block element and in-line element are reduced to a minimum amount. The div in the block element and the span in the in-line element are defined to give various extensions.
- The br element is defined so that you can break lines inside of the lists, or the cells in the table, also inside of the paragraph without ending the paragraph.
- Output format of the list and table can be specified by the attributes.
- The problem is that the content and style is mixed.

Hello! World

Simple Example of Transforming SimpleDoc into XSL-FO

Now, we show the simplest simple stylesheet that transforms SimpleDoc to XSL-FO as follows.

Source XML Document (Hello.xml)

```
<?xml version="1.0" encoding="UTF-16" ?>
<doc>
  <head>
    <title>Simple</title>
  </head>
  <body>
    <p>Hello World!</p>
    <p>This is the first<b>SimpleDoc</b></p>
  </body>
</doc>
```

XSL Stylesheet(Simple.xsl)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
                              xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes"/>

<xsl:template match="doc">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
        <fo:region-body   margin="20mm 0mm 20mm 0mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="PageMaster">
      <fo:flow flow-name="xsl-region-body" >
        <fo:block>
          <xsl:apply-templates select="body"/>
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="body">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="p">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="b">
  <fo:inline font-weight="bold">
    <xsl:apply-templates/>
```

```
  </fo:inline>
</xsl:template>

</xsl:stylesheet >
```

Generated XSL-FO

```
<?xml version="1.0" encoding="UTF-16"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
      <fo:region-body margin="20mm 0mm 20mm 0mm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="PageMaster">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>
        <fo:block>Hello World!</fo:block>
        <fo:block>This is the first
          <fo:inline font-weight="bold">SimpleDoc</fo:inline>
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

The above XSL-FO is formatted/displayed as follows.

Hello World!
This is the first **SimpleDoc**

Stylesheet Structure

The above Simple.xsl and XSL-FO show the following facts:

- A stylesheet is a set of templates. The descendant of the root element, xsl:stylesheet consists of xsl:template elements. Each xsl:template is applied so that the xxx tag of the source XML document may be processed by match="xxx".
- Formatting objects and the XML source text in each template are output to result XSL-FO tree. Then, templates that match to the descendant elements are called by an instruction of xsl:apply-templates.

XSLT processor loads the source XML document, starts processing from the root node. It finds the templates that match each node, and processes them as described in the templates. The processor processes child elements recursively, continues until the processor returns to the root element and there are no more templates to be processed.

Block Elementand Inline Element

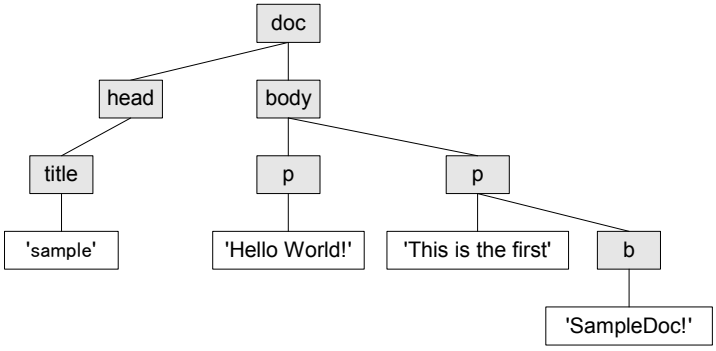
Please note how the XSL stylesheet maps block elements and inline elements in source element to formatting objects.

- In the stylesheet, p elements are transformed into fo:block objects, b elements are transformed into fo:inline objects. The base of XSL-FO transformation is to map the elements of the source XML document to either fo:block elements or fo:inline elements according to the layout instruction.
- The elements that intend to break lines by the end tag normally can be mapped to the fo:block objects. the elements of which the end tag do not intend to break lines can be mapped to the fo:inline objects. The attributes of source

elements specify properties of formatting objects. In this case, the b element means emphasis and property of the inline object generated from b is specified as bold.

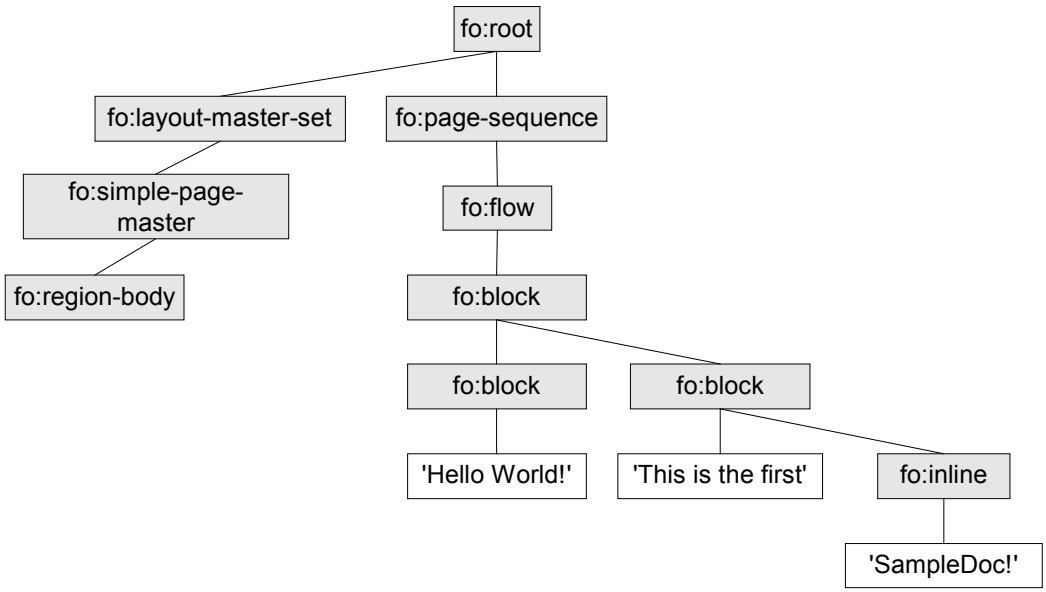
XSL-FO Tree Structure

Next, please pay attention to the XSL-FO tree structure. The following illustrates the structure of the XML document.



Hello.xml Tree Structure

In contrast, the tree structure of XSL-FO is as follows. The root of the XSL-FO tree is fo:root, which has two children, fo:layout-master-set and fo:page-sequence. Fo:layout-master-set defines the page layouts and fo:page-sequence has a flow of contents arranged in in pages.



XSL-FO Tree after XSL Processing

fo:layout-master-set that defines the page layouts should precede fo:page-sequence(preceding-sibling)that is an actual content of pages. XSL processor processes the XML source document from the root element, seeking the templates (xsl:template) to be matched. Therefore, **fo:layout-master-set element should be an output from the template that processes the root element of the XML source document.** In this case, <xsl:template match="doc"> takes this processing.

Shamkant B. Navathe

Evolution of Data Modeling for Databases

The discipline of data modeling initially became established because it provided a way for specifying the structures of data* in actual file systems followed by database management systems (DBMSs). This led to the introduction of the network and the hierarchical models in the 1960s exemplified by the DBMSs called Integrated Data Store (IDS) of Honeywell (network model) and Information Management System (IMS) of IBM (hierarchical model). The relational model was then proposed as a mathematical basis for the analysis and modeling of data [16], providing data independence and addressing a variety of problems related to databases. It also provided a way for addressing redundancy as well as estimating the goodness of database structures in a formal way. The relational model made it possible to deal with integrity constraints, security schemes, distribution of data, and replication of data which could not be rigorously specified and analyzed previously.

The focus has subsequently shifted to modeling data as seen by the application and the user. Basic data abstraction concepts of classification, generalization, aggregation, and identification were applied in different combinations and different degrees to produce a plethora of "semantic" data models in the late 1970s and early 1980s. This article traces this evolution of data models and discusses the recent developments that have dominated the commercial practice of data modeling: the entity-relationship, the functional, and the object-oriented approaches. The article concludes with an overview of the current areas such as modeling of dynamic, active databases, and knowledge discovery from databases.

Data modeling benefited immensely from developments in knowledge representation and terminological reasoning, and new models such as CANDIDE [10] are springing up as a result of the marriage between these two areas. Certain developments have dominated the commercial practice of data modeling: the entity-relationship [14], and the binary approach called NIAM [37] are two examples. The functional approach was exemplified by the DAPLEX model [34] and is having an impact on object models coupled with functions such as the Iris model, now available commercially from Hewlett-Packard in their Open ODB system. A variant of the ER model called IDEF/1X (see [25]) gained a good degree of following in some government establishments. Recently, a major effort for standardizing the representation and modeling of parts data and designs under the name Product Data Exchange using STEP (PDES) [32] is under way. STEP is the ISO standard for the Exchange of Product model data. This has brought about the renaissance of data modeling which is being applied in diverse industries such as building and construction, electrical components, and architecture. Thus, the scope of data modeling now extends far beyond what it was in the early days of file-oriented systems; many organizations are embarking on corporate data modeling as a part of the strategic planning activity.

Since this issue of *Communications* is devoted to different aspects of modeling that include object-oriented analysis and modeling as well as the knowledge representation area, we will not dwell heavily on it. Our focus will be on the modeling of data as applied to the design of database structures. We will highlight the current trends in object-oriented modeling of data as well as modeling of active databases. Some of the outstanding problems for data analysis and schema design include bridging the gap between requirements specification and data modeling, integrating diverse modeling paradigms, reverse-engineering existing databases into a conceptual abstract representation, and combining the procedural and declarative aspects of data. We will not, however, be able to address these problems in this article.

Basic Definitions

In this section we define some basic terminology for the database area pertaining to data modeling. Interested readers may consult [8, 19] for detailed explanations of these terms.

A **data model** is a set of concepts that can be used to describe the

*The word Data will be used in singular throughout this article in keeping with the convention in database literature.

structure of and operations on a database. By *structure of a database* we mean the data types, relationships, and constraints that define the "template" of that database. A data model should provide **operations** on the database that allow retrievals and updates including insertions, deletions, and modifications. Note that we will use the term "data model" to refer to the discipline for modeling data in a particular way—one that provides the building blocks or the **modeling constructs** with which the structure of a database can be described. We will use the term **application model** to refer to the description of data for a particular database. For example, the relational model is a data model. The definition of a particular database for the personnel application at company X will be called an application model of that database which uses the relational model. Application analysts often refer to the latter as a data model, causing confusion.

In any application, it is important to distinguish between the *description* of the database and the *database itself*. The description is called the **database schema.** A database schema is designed for a given set of applications and users by analyzing requirements. It is described by using a specific data model that provides the modeling constructs in the form of a language syntax or diagrammatic conventions. In some tools or DBMSs, the data model is not explicitly defined but is present implicitly in terms of the features present. Schemas usually remain relatively stable over the lifetime of a database for most applications. For dynamic environments, such as computer aided design (CAD), or computer-aided software engineering (CASE), the schema of the product being designed may itself change. This is referred to as **schema evolution.** Most DBMSs handle a very limited amount of schema evolution internally.

During the process of database design, the schema may undergo transformation from one model into another. For example, the schema of the personnel database may be initially described using the entity-relationship data model in the form of an ER diagram. It may then be mapped into the relational data model which uses structured query language (SQL)—an emerging standard, to define the schema. The entire activity of starting from requirements and producing the definition of the final implementable schema in a DBMS is called **schema design.** The DMBS is used to store a database conforming to that schema; it allows for database **transaction processing** in which a transaction is a unit of activity against the database that includes retrieval and update operations against the database. A transaction must be performed in its entirety, leaving a "permanent" change in the database or may be aborted, performing no change at all.

The actual database reflects the state of the real world pertaining to the application or the "miniworld." It must remain in conformity with the miniworld by reflecting the actual changes taking place. The data in the database at a particular time is called the "database instance" or the "database state." The actual **occurrences** or **instances** of data change very frequently as opposed to the schema, which remains static.

A data model in the database parlance is associated with a variety of languages: data definition language, query language, data manipulation language, to name the important ones. The **data definition language (DDL)** allows the database administrator or database designer to define the database schema. The DBMS has a compiler to process the schema definition in DDL and to convert it into a machine-processable form. This way, a centralized definition of the application model is created, against which a number of applications can be defined. **Data manipulation language (DML)** is a language used to specify the retrieval, insertion, deletion, and modification of data. DMLs may be divided broadly into two categories: declarative and procedural. The former allow the user to state the result (of a query) that he or she is interested in, whereas the latter require one to give a procedure for getting the result of the query. The nature of the language also depends on the data model. For example, while it is possible to have a declarative language for a model such as the relational model, the language for the network data model is "navigational," requiring the user to state how to navigate through the database, and thus is inherently procedural. Either type of language may be used in a standalone interactive fashion as a **query language.** Languages for data models can also be distinguished in terms of whether or not they are record-at-a-time or set-at-a-time. Record-at-a-time processing requires an elaborate control structure typically provided by a **host programming language** within which the DML commands or verbs are embedded. Set-oriented processing regards data as sets of elements (e.g., sets of tuples in the relational model) and provides for operators that apply to these sets, generating new sets. There is now a movement toward providing languages which seamlessly integrate the capability to provide general-purpose computation and special-purpose data manipulation against the database in a single language. These are called **database programming languages (DBPLs)** [6].

Scope of Data Models

In the traditional sense, data models used for database schema design have been limited in their scope. They have been used to model the *static* properties of data including the following:

Structure of data: The structure is expressed in terms of how atomic data types are aggregated into higher-order types. Furthermore, the models express relationships among these aggregates. The early models tended to be record-oriented, the basic aggregate struc-

ture being a record type consisting of data element types or field types. The database schema consists of a number of record types that are related in different ways. The hierarchical data model organizes the record types in a tree structure, whereas the network data model organizes a database schema with the record types as the nodes of a graph. The limitations of the record-based view of data modeling are discussed in [23]. The relational model introduced a set-oriented view of data modeling [16], and currently, the object-oriented view which structures a database in terms of objects and interobject interactions is gaining popularity [3]. We discuss these approaches in greater detail later in this article.

Constraints: Constraints are additional restrictions on the occurrences of data within a database that must hold *at all times.* Data model constraints serve two primary goals:

- Integrity: Integrity constraints are the rules that constrain the valid states of a database. They arise either as properties of data, or as user-defined rules that reflect the meaning of data.
- Security and protection: This applies to restrictions and authorization limitations that are applied to a database to protect it from misuse and unauthorized usage.

Constraints can be visualized at different levels:

a) **inherent constraints** pertain to the constraints that are built into the rules of the data model itself. For example, in the entity-relationship model a relationship must have at least two participating entity types (depending on the variant of the model used, the same entity type may be used twice).

b) **implicit constraints** are constraints that can be specified using the DDL of a data model to describe additional properties. They are expected to be automatically enforced. An example is a mandatory participation constraint in the entity-relationship model, which states that a specific entity must participate in a particular relationship.

c) **explicit constraints** are application-dependent constraints that specify the semantic constraints related to an application. These are the most general and difficult constraints to specify in full detail. There is a general trend to capture as much "application behavior" information within a database as possible in order to capture it in a central place. The 4GL movement is aimed at capturing this application semantics at a high level. Today's DBMSs are not equipped to handle such constraints easily, however.

Another dimension of constraint modeling is to capture state transition rather than just static state information. This gives rise to **dynamic constraints** which are stated in terms of what types of changes are valid on a database. An example is: "the salary of an employee can only increase." Both static and dynamic constraints allow us to define whether a database is consistent. Whereas static constraints can be evaluated on a "snapshot" of a database to determine its validity, the dynamic constraints are much more difficult to enforce, since they involve blocking/preventing a state transition at "run-time."

Other parameters of data models: A data model for a database may also specify some additional details relevant to the use of the data. One possible feature is **"distribution parameters."** These relate to the fragmentation of data in terms of how data is stored as fragments. In the relational model it is customary to refer to "horizontal" and "vertical" fragments. The former contain subsets of the data occurrences of a relation that meet some predicate condition, while the latter refer to a subset of the attributes of data for the whole relation. In a relation called ORDERS, each horizontal fragment may contain orders that are shipped from one warehouse, whereas ORDER may be vertically fragmented into shipping information and billing information. **Security** is another feature that may be built into a data model at different levels of granularity. Yet another feature is **redundancy,** which is hard to model explicitly; it may be captured in the form of specification of explicit copies or overlapping data. A model must allow for specification of features such as **keys** which uniquely identify data; it may also have a way to specify physical parameters such as **clustering** or **indexing** (e.g., B+ tree index on a certain field) as a part of the application model specification.

Views: In database modeling a **view** is a perceived application model as defined by a user or an application group. A view is another mechanism by which an application can record its specific requirements, in addition to the explicit constraints mentioned previously. Most data models provide a language to define views: it may be coincident with the query language, whereby the result of a query is defined as the view. Typically, a view is constructed when there is a request to retrieve from it, rather than "materializing" the view because the latter creates redundant data.

Toward Joint Data and Application Modeling

To give adequate support for the modeling of dynamic application environments, another school of thinking combines the functional analysis, which is typically the focus of software engineering during information system design, with conventional data modeling [11, 26], giving rise to a joint analysis and modeling of application and data requirements. We have also advocated this view during conceptual database modeling in [8]. In terms of the present issue of *Communications,* the preceding approach seems most relevant and significant. A top-down structured design methodology for both data and process modeling using extended ER and data flow diagrams respectively is