Paper DM01

New Technologies for Delivering Data to Internal and External Clients

Sigurd W. Hermansen, Westat, Rockville, Maryland

ABSTRACT

Back in the Big Iron era, programmers captured, stored, and delivered data in the same form. Files were all we knew. Now we have visual representations: Web data entry forms, MDBB's, Web page browsers, and electronic codebooks. What we consider the same 'data' have different shapes and structures in container objects than in an RDBMS or in an XML stream. So, how do we get from forms to data? How do we serve data to internal or external clients? To explore these questions we trace a set of data from capture to database to delivery. Blaise(R), Oracle Clinical(R), Web server pages, and SAS(R) FS/AF/SCL forms illustrate alternative DE programs. 'Flat files', 'drill downs', RDBMS schema, and tagged fields in data streams serve as examples of data products. We will illustrate methods that preserve essential data as they pass through a series of transformations via several scenarios.

INTRODUCTION

Displays of data provide visual cues to help humans comprehend them. Viewers of data displays do not typically distinguish data from visual cues when they look for information. A simple paper form defines data items or fields:

```
Person name:_____ ID _____ #dogs _____
dogs:     1.        breed____ age_____
          2.        breed____ age_____
                    ……..
          Infraction ___      dog __    date _____
                     ___      dog __    date _____
                       ……..
```

Viewers understand intuitively what to enter in each field. Good form design promotes reliably intuitive understanding of the pragmatics of data entry and display.

Computers have none of the visual intuition that viewers have. Sequencing, implicit typing, and spacing do not translate directly into bits and bytes. Precise identification of data items requires an explicit indexing scheme. Mapping from data displays to computer storage has to be done under the control of complex and sophisticated programs.

Developers of data entry forms (DEF) and data display forms (DDF) rarely start programming these data objects from scratch. A number of programming environments provide automated support for development of DEF's and DDF's. Each programming environment supports one or more databases (DB's) under the hood.

Database programmers specialize in mapping and transforming data from DEF to DB and from DB to DDF. In a brave new world of enterprise, distributed, federated, and Web databases, cross-platform and cross-database development has become the norm, not the exception. Examples in the next sections illustrate how data morph as they migrate from DEF to DB and from DB to DDF.

DATA ENTRY FORMS (DEF'S)

Developers of a DEF have target applications in mind. An automated data collection instrument presents visual cues and slots for entering data into forms. Depending on the design and capabilities of the system supporting the DEF, it may also display data that have already been stored in a DB. That requires an ability to capture identified information first, then to transmit a query that selects data from a DB. The DEF queries the database to validate entries, and to insert or update a DB.

DEF: BLAISE SCREENS

Blaise is a widely-used software system for computer-assisted interviewing (CAI) and survey processing. A DEF targeted at CAI must replicate the basic process of displaying a questionnaire form and using a pencil to enter data in prescribed slots. Further, it must implement 'skip patterns' in survey instrument by displaying one screen or another depending on the response to a lead-in question. (eg., Q29: How many dogs do you own? If 0, skip to Q35; else answer Q30-Q34 for each dog.) We can picture this pattern of responses as a row of columns that bulges into a table to accommodate data on one or more dogs:

```
Column[person1]:    #dogs _____
                |   dog i=1:  breed____ age_____
table:          |   dog i=2   breed____ age_____
                | ….
                | Inf_i=1:___          dog __    date _____
table:          | Inf_i=2:___          dog __    date _____
                |              ……
Column[person2]: ……
```

Whether clients might wish to have data delivered in the same form depends on intended uses of the data.

DEF: ORACLE CLINICAL CASE REPORT FORMS (CRF'S)

Tight control of reporting of adverse reactions to drugs has led to rigid standardization of DEF's designed to support clinical trials. Each trial accumulates an unbounded stack of case event reports from medical centers. A simplified event form captures column values in each row of CRF data:

```
protocol: _____
  Person: _____
     Dog: _____
       event_date:_____
       event_type: _____
   event: _____
eg. (12,Jones,3,12/18/02,INF,X3243)
```

The first five data items index each event. The order of these index values does not matter. The event_type has the same role as a column name in a row of patient events. Expanding the domain of the event_type allows the form to accommodate new types of events without modifying the DEF (or the DB structure underneath the DEF).

DEF: SAS/AF® FRAMES

The SAS® System V8 SAS/AF product serves as a convenient example of a visual object toolkit that programmers can use to create simple or complex DEF's. Visual objects appear as displays or parts of displays on a computer screen, or control how displays 'behave'. Objects have properties. Programmers control the behaviors of objects by sending messages to detect and change property settings.

SAS/AF objects bind values entered on a DEF object to object property names. A programmer assigns instances of properties to data elements in a database.

DEF: HTML FORMS

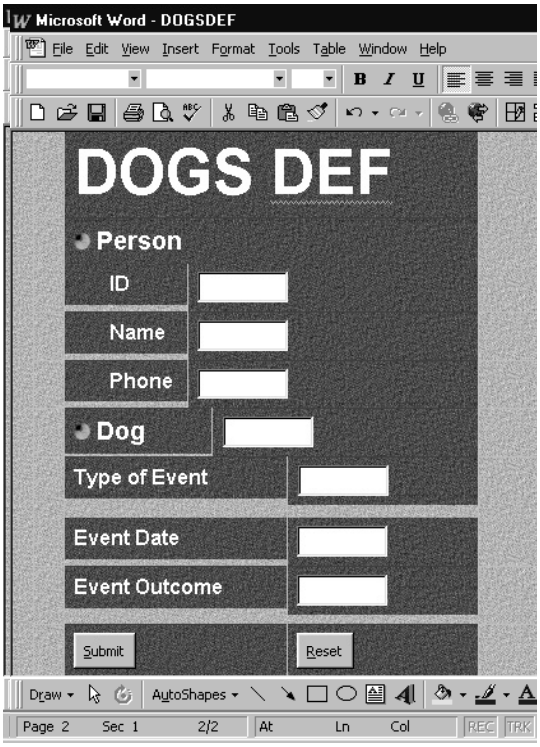
The HTML protocol supports DEF's in Web pages. The <**FORM**> and <**INPUT**> tags identify different types of input fields in an HTML DEF and bind the value entered in the ID slot on the HTML form to a variable label (Id):

<**FORM**>

……

```
<TR><TD WIDTH="3%" VALIGN="TOP"
HEIGHT=31><P></P></TD>
<TD WIDTH="13%" VALIGN="TOP" COLSPAN=2 HEIGHT=31>
<B><FONT FACE="Arial"><P>      ID</B></FONT></TD>
<TD WIDTH="31%" VALIGN="TOP" COLSPAN=5 HEIGHT=31>
<FONT FACE="Arial"><P>
<INPUT TYPE="TEXT" MAXLENGTH="50" NAME="Id">
</FONT></TD>
<TD WIDTH="52%" VALIGN="TOP" COLSPAN=4
HEIGHT=31><P></P></TD>
</TR>
……..
```

Which displays in a Web browser as



Web browser programs (Netscape®, MS Internet Explorer, Opera, Mozilla, etc…) support HTML DEF's and attach lists of label-value pairs to hyperlinks.

DEF: SERVER PAGES
Although open and flexible, HTML forms seem designed primarily to support transfers of parameter values or other limited lists of data items. Java Server Pages (jsp) and Active Server Pages (asp) provide alternatives to HTML forms that support enhanced forms, better data validation, and faster and more secure data transfer methods. Support for server pages comes largely from servers.

Data for delivery to clients often originate from more than one DEF. Different DEF's transfer data to central databases. The visual format of each DEF has to map to a common database structure.

DATABASES (DB'S)

Date makes a point of distinguishing a 'database', a logical framework for data and the data that populate it, from a database system that houses databases, whether DB2, Oracle, SQLServer, Sybase, Informix, Ingres, or, yes, MySQL, and those database systems outside the relational database system (RDBMS) mold, such as XML databases. In the same sense, twelve ounces of cola does not become 'bottle cola' when bottled or 'can cola' when canned. In fact, to their discredit, database system developers put additives in databases that blend logical database and implementation features, and add to the difficulty of porting databases to other database systems.

Database systems provide tools for database design and for maintaining data integrity, but good database design and useful database systems do not come in a box. Client requirements for data deliveries often include irreconcilable differences. For example, a requirement to deliver data as a 'flatfile' may force database programmers to scrunch many interrelated dimensions of observations into a two-dimensional structure. The next sections present alternative data models and their strengths and limitations. Each section includes a brief discussion of data transfers to and from each data structure.

DB: FILE
A general layout of a file uses field names and descriptions to index and model a DB. Repeating blocks (records) in a file map to field names according to a scheme:
field-indexed repeating groups data model
group[dog]:=(breed,age)
record[person]=ph#,group[dog1],group[dog2],
eg. Jones,555-1212,dog1(cocker,3),dog2(mutt,2)

Each record ends with a line marker that separates one record from another. An alternative data structure stores the same data in repeating records:
record[person]
record[person,dog1]
record[person,dog2]
…..
eg. Jones,555-1212
cocker,3
mutt,2
……

The SAS System INFILE and INPUT statements and the DBMS/COPY product provide a truly mature and very useful technology for data capture from these and other special file formats generated by DEF's. Blaise exports field-indexed data and SAS INPUT statements to read them (among the several options Blaise offers for exporting data).

DB: MDDB
The two dimensional index [person,dog] points to a row of data in a repeating record structure. In a multidimensional database (MDDB), a full index points to individual data items, and partial indexes point to rows or tables
(eg. value[person,dog,event_date,event_type]).
The SAS MDDB product and other specialized systems map data from DEF's or intermediate data sources. An MDDB extends an old technology, multidimensional arrays, to database object classes with associated constructor methods. An MDDB often contains summary data as well as details.

DB: XML
The hierarchical XML data model extends data value tagging in Web documents to DB indexing. XML interprets a hierarchical structure of tags as a multidimensional index:

```
<?xml version="1.0" ?>
…….
<PERSONS ID="111" NAME="Jones">
  <PETS ID="222" BREED="mutt">
            <FACT>
                    <PETID>222</PETID>
                    <EVDT>12/18/02</EVDT>
                    <EVTYP>INF</EVTYP>
                    <EVNT>2203</EVNT>
            </FACT>
            <FACT>
                    <PETID>222</PETID>
                    <EVDT>12/18/02</EVDT>
```

```
                    <EVTYP>VAC</EVTYP>
                    <EVNT>distemper</EVNT>
                </FACT>
            </PETS>
</PERSONS>
```

ÖÖÖ

The hierarchical XML data model embeds metadata in tags and forces a somewhat arbitrary distinction between data labelled as attributes and tagged data elements. The metadata tags make it almost impossible to display or search an XML document without mapping it to another format.

Fortunately, SAS provides thel XML Mapper application in V9 to assist in designing an XMLPath map specification for specific forms of XML documents:

```
<?xml version="1.0" ?>

<!-- 2002-12-10T02:39:43.786 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XMLAtlas, Version 9.0.1 -->

<SXLEMAP version="1.1" name="SXLEMAP" description="">

 <TABLE name="PERSONS">
  <TABLE-PATH>/PERSONS</TABLE-PATH>

  <COLUMN name="ID">
   <PATH>/PERSONS/@ID</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>3</LENGTH>
  </COLUMN>

  <COLUMN name="name">
   <PATH>/PERSONS/@name</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>5</LENGTH>
  </COLUMN>

 </TABLE>

 <TABLE name="PETS">
  <TABLE-PATH>/PERSONS/PETS</TABLE-PATH>

  <COLUMN name="PersonID">
   <PATH>/PERSONS/@ID</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>3</LENGTH>
  </COLUMN>

  <COLUMN name="ID">
   <PATH>/PERSONS/PETS/@ID</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>3</LENGTH>
  </COLUMN>

  <COLUMN name="breed">
   <PATH>/PERSONS/PETS/@breed</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>4</LENGTH>
  </COLUMN>

 </TABLE>

 <TABLE name="FACT">
  <TABLE-PATH>/PERSONS/PETS/FACT</TABLE-PATH>

  <COLUMN name="PersonID">
   <PATH>/PERSONS/@ID</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>3</LENGTH>
  </COLUMN>

  <COLUMN name="PetsID">
   <PATH>/PERSONS/PETS/@ID</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>3</LENGTH>
  </COLUMN>

  <COLUMN name="EVDT">
   <PATH>/PERSONS/PETS/FACT/EVDT</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>8</LENGTH>
  </COLUMN>

  <COLUMN name="EVTYP">
   <PATH>/PERSONS/PETS/FACT/EVTYP</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>3</LENGTH>
  </COLUMN>

  <COLUMN name="EVNT">
   <PATH>/PERSONS/PETS/FACT/EVNT</PATH>
   <TYPE>character</TYPE>
   <DATATYPE>STRING</DATATYPE>
   <LENGTH>4</LENGTH>
  </COLUMN>

 </TABLE>

</SXLEMAP>
```

The SAS XML engine uses the XMLPath map as a guide for parsing an XML document:

```
filename inxml "\\..\person.xml";
filename inmap "\\...\person.map";
libname inxml xml xmlmap=inmap;

proc contents data=inxml._all_;
run;
data persons;
        set inxml.persons;
run;
data pets;
        set inxml.pets;
run;
data fact;
        set inxml.fact;
run;
filename _all_ CLEAR;
libname _all_ CLEAR;
```

The SAS XML engine decomposes the hierarchical relation Persons – Pets – Facts into three tables (datasets) logically linked by person and pet ID's. The three tables retain all of the essential information in the XML document.

DB: RELATIONAL DB (RDB)

In contrast to field-indexed or pointer-indexed DB's, a relational database uses column names as column indexes and columns of key data values as row indexes. A normalized relational data model represents relations among different dimensions of data as linked

tables.

Normalized RDB data model

Persons	Dogs	Infractions
ID	ID	dogID
Name	breed	type
	Age	date
	ownerID	

A normalized RDB has an innate property that recommends it as a deliverable to clients: The normal form of the database supports reshaping of data into any structure the client may want (though not necessarily from a reshaped set of data back to the original relational scheme).

DB: DIMENSION-INDEXED FACTS SCHEME (DIFS)

In databases that contain data with many dimensions, relations among data elements may number in the hundreds. A fully normalized database divides each relation into a separate table and soon reaches a level of complexity that complicates data entry and reporting, and makes modifying the database a nightmare. Further, few internal or external clients have the time or expertise to select and combine data into useful datasets. While a dimension-indexed DB scheme has the same design as a relational database for the major dimension of a database (household, person, company, etc.), a single 'fact' or entity-attribute table, indexes interrelated data elements that might appear in many different tables in a relational database. Others have called the same structure an Entity-Attribute-Value (EAV) data model because it combines entity identifiers and attribute labels in a composite key that serves as a natural index to data items.

normalized DIFS data model

Persons	Pets	Events
ID	ID	petID
Name	breed	typePet
	birthDate	date
	ownerID	eventType
		event

This model combines a normal form with a structure that accommodates new classes of data. Data warehouse architects favor variants of the DIFS data model ,called star or snowflake schema, for databases that house many repetitions of fact records.

DATA DISPLAY FORMS (DDF'S)

Deliveries of data to clients may include DB's, DDF's, and metadata. In some instances, clients may want data subsets in different database structures. Deliveries of public-use datasets or research data also require access to metadata and detailed codebooks/data dictionaries. DDF's map selected contents of databases into forms that build in visual cues for viewers.

DDF: REPORT OBJECT GENERATORS

A popular, generic report generator, Crystal Reports, belongs to a class of programs that package formats and data into a report object. It constructs objects that map DB contents to a DDF.

Database programmers set up a library of reports that clients can select and run. Some prepackaged reports prompt users for parameter values. A typical report generator builds queries of one or more databases. Changes in database contents change the yields of queries, and thus change the contents of reports.

DDF: SAS/GRAPH

SAS/Graph operates much the same as a report object generator. It generates graphic objects and stores them in special graphics catalogs. Clients with access to the SAS System can edit and reformat graphic objects in SAS/Graph catalogs.

DDF: HTML/XHTML/PDF DOCUMENTS

Document mark-up languages automatically map data from databases to Web pages that give clients static and dynamic DDF's. Arrays of hyperlinked values have the same function as MDDB drill-down DDF's and marginal thumbnail indexes.

The ODS and Mark-Up contexts of the SAS programming environment maps results of SAS PROC's to document types that have extended support for an embedded mark-up language (HTML, RTF, XML, pdf). Stylesheets, templates, and schema bring visual elements effectively into dynamic mapping of database elements to DDF's.

WEB SERVLETS AND SERVER PAGES

Servlets and server pages give Web page developers convenient access to DDF's and supporting DB query languages. This example (developed by James W. Cooper) illustrates the simplicity and flexibility of Java servlet technology:

```
<HTML>
<TITLE> Servlet Test </TITLE>
<BODY> <H1> Servlet Test </H1>
<FORM ACTION="/servlet/HiYou"
METHOD="POST">
<INPUT TYPE="text" NAME="name" SIZE="20">
<INPUT TYPE="submit" VALUE="Submit">
</FORM>
</BODY></HTML>
```

The servlet produces a form into which a viewer can enter, for example, a report parameter value:

Servlet Test

Submit

Server pages support DDF's much as they support DEF's. Embedded queries on Web pages supply data that populate DDF's. Web and internet servers (predominately Apache and IIS) offer alternatives to MS Windows dynamic-link libraries and supplant much of the functionality formerly provided by operating systems.

SAS in Version 9 catalogs the output of SAS procedures, including PRINT, REPORT, and TABULATE, in DDF object catalogs much like SAS/Graph catalogs graphic objects. Object libraries combined with new interactive technologies open up new possibilities for delivering DDF object catalogs to clients. Increasingly, internal as well as external clients are asking database programmers to build options for selecting, reformatting, and displaying information in databases.

CONCLUSION

No one structure and organization of data meets all the needs of all clients. Visual forms display data in fundamentally different arrangements than one needs for storage of data on permanent media. New technologies make mapping and reshaping of data easier and better at preserving information in data. Web technologies have kicked off a revolution in data collection, database programming, and presentation of information. We have selected a small sample of emerging technologies to present. The rapid

expansion of computer systems now underway suggests that new technologies are emerging at accelerating rates.

SAS solutions have played important roles in discussions of mapping data from visual forms to databases and back to visual forms. Emphasis on SAS tools for database programming not only suits the forum, it also aligns well with the realities of database programming.

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

ACKNOWLEDGEMENTS

Colleagues at Westat, especially Kellar Wilson, Francis Harvey, Karin Davis, and James Kuan, contributed to content and the style of presentation. Mike Rhoads and Duke Owen reviewed the paper and made valuable editorial suggestions. Francis Harvey guided me through the twists and turns of SXLE architecture. The author takes full responsibility for errors or omissions that remain.

REFERENCES

Date, C.J. and Hugh Darwen, **Foundation for Object/Relational Databases: the Third Manifesto** (1998) Addison-Wesley (see p. 144).

Freibel, Anthony, '<XML> at SAS®- A More Capable XML Libname Engine', Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference, 27, (2002) 179.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sigurd W. Hermansen
Westat
1650 Research Blvd.
Rockville, MD 20850
(301) 251-4268
hermans1@westat.com