Integrating Oracle 10*g* XML: A Case Study

Coleman Leviter, Arrow Electronics

Introduction

XML DB was introduced in Oracle 8i. In Oracle 11g, XML DB reached a new maturity level, providing high-performance with native XML storage and retrieval technology. The W3C XML data model is fully immersed into the Oracle Database.

How is XML DB used in a project? What must one know using XML DB? In this discussion, we will address these issues and design criteria one might consider using XML in a project.

XML or Extensible Markup Language means the language can grow as required. You may include an XML declaration line for the first line (`<?xml version="1.0"?>`). You may define your own elements or tag fields in the body of an XML document. As long as the sender and receiver agree on the format of an XML Document, there is complete flexibility for its construct.

Throughout this paper, we will present several examples of XML documents. We will present the project as well as XML's involvement. We hope the readers will become familiar with XML by finding examples, modifying them and using them in their own schema. The WEB contains a large amount of XML material. The last Google count (for XML) was 420,000,000 hits! For those beginning an XML project, this establishes a good starting point.

Project Overview

The project uses XML for communications between a Warehouse Management System (WMS) and a Transportation Management System (TMS). The following reasons are given for using a TMS:

- Leave transportation decisions to a third party system which has the expertise to add, drop and manage shipping carriers
- Include rate shopping and efficient route selection
- Improve efficiency in the warehouse by removing several independent workstations supplied by different carriers leaving all transportation management decisions in one workstation

When product ships from Arrow Electronics, Inc., the customer may decide the mode of transportation. When the choice is "best way", a semi manual process selects a carrier at the least cost. Human intervention is involved. Different shipping workstations from various carriers (i.e. DHL, BAX, LTL, Watkins, etc.) provide choices at ship time. Shipping department managers make "best way, cheapest cost" decisions.

As business grows, Arrow Electronics, Inc. desires to increase efficiency in its shipping department. An integrated and automated Transportation Management System (TMS), which connects to its existing Warehouse Management System (WMS), will provide new shipping efficiencies over the previous semi manual and time consuming system.

The steps for shipping product are:

1. Pack station operator prepares product and carton(s) for shipping
2. Operator reviews shipping instructions and determines if further shipping labels and documents are required
3. If further labels and documents are required, the operator walks to the shipping carrier's stand alone station (i.e. DHL, FEDEX, BAX, LTL, etc.) and produces the necessary information, manually typing in all the data
4. Material is now ready for pick up by the shipping carrier

By using the TMS, we eliminate steps (a) and (b), which are labor-intensive operations. All shipping labels and pertinent documentation print at one station.

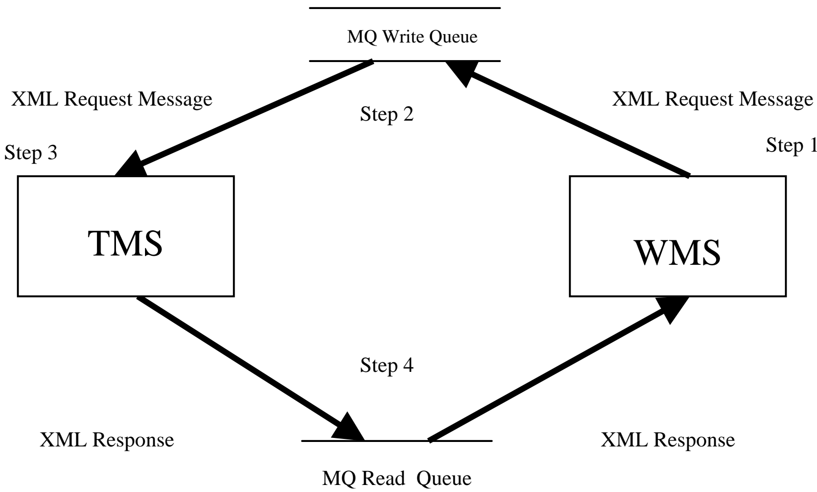
Presented is the WMS/TMS communications data flow:

Step 1) WMS Application issues an XML Request Message (XML Document) to MQ Write Queue
Step 2) TMS, listening for messages, receives the XML Request Message

Step 3) TMS formulates its answer and issues an XML Response Message to MQ Read Queue

Step 4) WMS, waiting for the response from Step 1, reads the XML Response Message

Figure I – WMS/TMS XML Document Data Flow



Message Comparison

Why use XML for messaging? To answer that question, let us explore an alternate method of data transfer.

One predominant method of transferring messages between two computers (or processes) is Binary Messaging. This method continues to be in major use with many legacy computers and embedded firmware systems.

Data may be compressed, so encoding and reassembly must all work in concert. If a transmitted data steam contains 256 bits, the receiving side must also be in alignment and decode those same 256 bits. All data is position dependent.

When implementing this method, maintenance costs may run very high because of the volume of software that must be managed on the sender and receiver sides. Troubleshooting adds to the cost as well.

Describing a system's efficiency is the ratio of output to input. Therefore, the efficiency of messages between two computers is shown as:

$$\text{Efficiency (\%)} = \frac{\text{Output}}{\text{Input}} \times 100 \quad \text{or} \quad \frac{\text{Message Data}}{\text{Message Data + Overhead Data}} \times 100$$

Sales Complete Message (SCP)

Field Name	Field Type	Size	Bit Position
Overhead Data	Alphanumeric	32	1 - 32
Message Code	'SCP* '	4	33-36
Entering Location	Alphanumeric	3	37-39
Sales Number	Alphanumeric	6	40-45
Version Number	Alphanumeric	2	46-47
Cartons for Shipping	Numeric	2	48-49
Shipping Charges	Alphanumeric	9/2	50-58
Date (YYMMDD)	Alphanumeric	6	59-64
Carrier	Alphanumeric	16	65-80
Carrier Number	Alphanumeric	19	81-99
Shipment Weight	Alphanumeric	4	100-103
Number of Orders	Alphanumeric	2	104-105
Ship Code	Alphanumeric	2	106-107
Spare 1	Alphanumeric	3	108-110
Spare 2	Numeric	7	111-117
Spare 3	Alphanumeric	4	118-121
Spare 4	Alphanumeric	1	122-122
Spare 5	Alphanumeric	1	123-123
End	Alphanumeric	27	124-150

Table I – Typical Binary Interface Data Layout

Using the sample binary data layout from Table I we have:

118 bits (pos 33- 150)
---------------------- x 100 = 79% efficient
(Message data) 118 bits + 32 bits (overhead data)

Let us look at an excerpt from a typical XML Document (447 bytes):

```
<SHIPREQUEST>
  <!-- this is 182, svia J4 -->   ←--------------------- comment
  <SHIPMENTINFO>
    <FACILITYCODE>RNO</FACILITYCODE>
    <TARGETDATE>05/10/2007</TARGETDATE>
     …                  ^-------------------------  typical data…
     <NUMBEROFSKIDS/> ←----------------------- empty element
    <BOLDESCRIPTION>ELECTRONIC COMPONENTS</BOLDESCRIPTION>
    <BOLCLASS>85</BOLCLASS>   ^-------- typical data
    <BOLCOMMENTS/>
  </DOCDATA>
</SHIPREQUEST>
```

Figure II - Excerpt from an XML Message

Using the Efficiency Formula from above we have:

447 Bytes Message Data (Figure II)
--- x 100
447 Bytes Message Data + 3437 Bytes XML Element Definition (= 3884 total bytes)

= **12 % efficient**

Therefore, using an XML data message results in 12 % efficiency, while using binary data transfer results in 78 % efficiency. It is obvious that the messages are quite different, but an important observation is the overhead ratio required to send data. XML messages far exceed binary data messages, mainly due to the verbose nature of the element names in the XML Document.

If the efficiency of an XML data message does not fare well compared to a binary data message, then why use XML data messaging? Here are arguments for and against:

XML Arguments For Usage

- It is platform and system independent i.e. it can work on any computer.
- It allows us to define our own tags thus making your data content more clearly.
- XML has adopted a standard, **ISO 10646** also known as Unicode, which is a framework to encode characters and it will support most languages, thus not forcing people to use English for coding.
- Software can be developed to increase efficiency, that is, encode the element tags on the transmission as well on the receiving side.
- The code is easy to understand even for those people who do not have any prior knowledge.
- XML DB has been available with Oracle 8i and up
- XML is self-describing. For example it is obvious from <lastname>Doe</lastname> that this represents a name.
- Bandwidth issues can become non existent using data compression techniques

XML Arguments Against Usage

- It requires a wide amount of bandwidth. (This should improve in the future using data compression techniques)
- It may require extensive processing time to decode
- Only newer software will be able to read and understand XML. It may become costly and time consuming to retrofit legacy code with XML

A good design decision rests with one's ability to analyze a problem and choose the proper tools. The same applies when selecting XML or another data communications method. In the previously described application, the new TMS system already used XML.

Our system used Oracle 10g in which XML DB was available. So the decision was based upon the experience of the organization and product of a third party package (TMS) already using XML. Although the XML duty cycle is low, if the messaging frequency is low (which was in our case), XML is certainly a viable option. For example, if the application environment is limited to human interaction with a computer and waiting for an answer (i.e. credit card verification), XML inefficiencies would not appear to present a problem. If XML messages were used to guide a shuttlecraft into its docking station, perhaps too many messages would rapidly use up the bandwidth. Another example where XML data transfer might not work: a central station (monitoring burglar alarms) receiving video transmissions of an alarm of a potential intruder. With samplings of several seconds and several frames of a possible intruder in the area, the data transmission with compression may not exceed 250 Kbytes. This real time video example may not be a candidate for XML. However, this might be a viable candidate in the future using data compression techniques.

XML Communications

Our message queuing system for XML communications is IBM's WebSphere MQ Message Queuing, which is in use with many enterprise applications.

When we view XML Messages on MQ, we use a tool called IBM Tivoli CandleNet Portal. Here is an extract of a sample message using the tool:

```
Hexadecimal Data                        | Character Data
3C3F786D 6C207665 7273696F 6E3D2231 | *<?xml version="1*
2E302220 656E636F 64696E67 3D225554 | *.0" encoding="UT*
462D3822 3F3E3C53 48495052 4553504F | *F-8"?><SHIPRESPO*
4E53453E 3C505249 4E54464C 41473E4E | *NSE><PRINTFLAG>N*
3C2F5052 494E5446 4C41473E 3C504143 | *</PRINTFLAG><PAC*
4B414745 533E3C43 4152544F 4E4E554D | *KAGES><CARTONNUM*
4245523E 313C2F43 4152544F 4E4E554D | *BER>1</CARTONNUM*
4245523E 3C4D534E 3E333838 3C2F4D53 | *BER><MSN>388</MS*
4E3E3C54 5241434B 494E474E 554D4245 | *N><TRACKINGNUMBE*
523E315A 31323334 35363033 31323334 | *R>1Z123456031234*
39353234 3C2F5452 41434B49 4E474E55 | *9524</TRACKINGNU*
4D424552 3E3C4C41 42454C5A 504C3E37 | *MBER><LABELZPL>7*
45344134 31304430 41374535 33343433 | *E4A410D0A7E53443*
32333030 44304430 41354535 38343135 | *2300D0D0A5E58415*
45343334 39333133 33304430 44304135 | *E434931330D0D0A5*
45344334 38333032 43333035 45343635 | *E4C48302C305E465*
33304430 44304135 45353035 32333432 | *30D0D0A5E5052342*
43333432 43333235 45343635 33304430 | *C342C325E46530D0*
44304135 45343635 34333032 43333133 | *D0A5E4654302C313*
```

Figure III – Partial XML Document viewed using IBM Tivoli CandleNet Portal

The left side of Figure III shows the XML message in hexadecimal; the right side shows the XML message. The asterisks are not part of the data. Viewing XML data in this manner aids in isolating problems when the XML message is on its way to the receiver or arriving from sender.

XML Document Construction

Properly formed XML documents contain (except where noted) the following components:

- An optional declaration as the first line (generally, it is good practice to include the declaration, but it is not mandatory):

```
<?xml version="1.0" encoding="UTF-8"?>
```

where the mandatory first attribute identifies the XML version number and the optional second attribute is the encoding attribute which specifies to the XML parser what character encoding the text is in for translation into Unicode (Unicode is an industry standard allowing

computers to consistently represent and manipulate text expressed in any of the world's writing systems).

- An optional comments section:

```
<!-- this is a comment 182, svia J4 -->
```

A mandatory start tag and end tag as the root element:

```
<MAIN_TAG>         < ------ start tag

</MAIN_TAG>        < ------ end tag
```

And finally, a mandatory data element:

```
<Data_Section>this is a data
section</Data_Section>
```

So, at a minimum, one root element and one data element constitutes a properly formed XML document.

Putting it all together we have:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- this is a comment 182, svia J4 -->
      < MAIN_TAG >
<Data_Section>this is a data
section</Data_Section>
</ MAIN_TAG >
```

The XML document fits a hierarchical model as presented in the Figure IV:

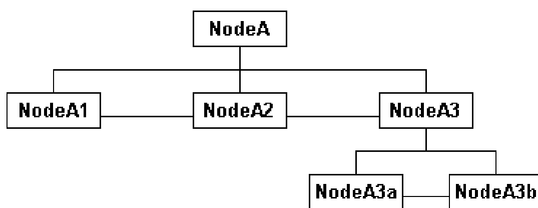


Figure IV – Hierarchical Model of an XML Document

where Node A is the root element and Node A1 … Node A3b represents the data section.

XMLType Column

To create a sample table containing an XMLType column, enter the following:

```
CREATE TABLE my_table (id number, xmlcol XMLTYPE);
```

The underlying type, XMLType is a CLOB, which enables storage of up to 4GB of data. Additionally, you may perform XPath queries on the XML Documents residing in the column. By simply defining the column (xmlcol) as a CLOB, XPath expression queries are not possible. When storing XML documents into an XMLType column, Oracle will raise an exception if the XML document is not properly formed. If you want to store the improperly formed document for later evaluation, depending upon its length, it may be stored in a CLOB type column.

When using XMLType columns in tables, it may be helpful to use the pragma `AUTONOMOUS_TRANSACTION. When a` subprogram is marked with this pragma, it is possible to perform rollbacks or commits without affecting operations in the parent transaction. Basically, this pragma works the same way as a sequence object.

XML Examples

In this section we will explore several XPath and SQLX (or SQL/XML) examples for shredding and creating XML Documents, respectively. Most of the code found in these examples is similar to code from the project. XPath or XML Path Language is a language for selecting parts of an XML document and computing values (strings, numbers, or Boolean values) based on the content of an XML document. We will produce several XML fragments using functions XMLElement(), XMLAgg() and XMLForest(). We will demonstrate document shredding using the XMLSequence() function and the EXTRACT method.

SQLX Document Construction

XMLELEMENT()

The simplest SQLX Query uses the XMLElement() function, which returns an
XMLType expression (XML fragment):

```
SELECT XMLELEMENT("Emp", 'jones')
employee FROM DUAL;

EMPLOYEE
```

```
----------------
<Emp>jones</Emp>
```

XMLForest()

This example, which uses XMLForest(), produces a fully qualified XML document:

```
SELECT XMLType.getclobval(
XMLELEMENT("trial_1", XMLFOREST(id_num,
two_cardinal) )
              ) as xmlexample1
FROM (select 1 as id_num, '1ST and 2ND'
as two_cardinal from dual)

<trial_1>
<ID_NUM>1</ID_NUM>
<TWO_CARDINAL>1ST and 2ND</TWO_CARDINAL>
</trial_1>
```

The XMLForest() function produces an XML fragment that contains a set of XML elements. XMLElement() uses the document fragment for children (or sub-elements) of its element, which in this case is tag field <trial_1>. XMLElement () returns a properly formatted XML document. XMLType is a dedicated XML datatype that is a CLOB type behind the scenes. It has a number of member functions available to make the data available to SQL. XMLForest's arguments are expressions or column names. XMLForest returns a concatenated XML fragment. In our example, id_num and two_cardinal are the alias names used in the subquery in the FROM clause.

XMLAGG()

The first example returns an ordered set using the XMLAgg() function:

```
SELECT XMLELEMENT("olin",
XMLAGG(XMLELEMENT("SALES_ORDER",
o.SORD_NUM||CHR(32)||o.sord_rel) ORDER BY
o.sord_num)) AS "sales order list"
FROM olin o
WHERE o.whse_code = 'PW'
AND ROWNUM < 6

<olin>
  <SALES_ORDER>3033920 09</SALES_ORDER>
  <SALES_ORDER>3372306 06</SALES_ORDER>
  <SALES_ORDER>3644585 02</SALES_ORDER>
  <SALES_ORDER>3662786 02</SALES_ORDER>
  <SALES_ORDER>3691640 02</SALES_ORDER>
</olin>
```

We have restricted the list to less than six elements. The XMLAgg() function returns an XML fragment in an XMLType by assembling XML fragments, with the option of XML element sorting.

The XMLAgg() function assembles all the XML elements into one XML document fragment. The outer XMLElement() function, incorporates the XML document fragment into its "olin" element as sub elements.

The next XMLAgg() example assembles the XML elements into one XML document fragment. However, this time an XMLElement() function returns an XML document fragment to its parent, an XMLElement() function. After the entire SQLX query is complete, the XML document fragment is assembled by the XMLAgg() function.

```
SELECT XMLELEMENT("olin",
XMLAGG(XMLELEMENT("SALES_ORDER",
XMLELEMENT("detail_info",o.sord_num||CHR(
32)||o.sord_rel) )))
AS "sales order list"
FROM olin o
WHERE o.branch_code = '91'
AND ROWNUM < 6

<olin>
  <SALES_ORDER>
    <detail_info>3033920 09</detail_info>
  </SALES_ORDER>
  <SALES_ORDER>
    <detail_info>3372306 06</detail_info>
  </SALES_ORDER>
  <SALES_ORDER>
    <detail_info>3691640 02</detail_info>
  </SALES_ORDER>
  <SALES_ORDER>
    <detail_info>3644585 02</detail_info>
  </SALES_ORDER>
  <SALES_ORDER>
    <detail_info>3662786 02</detail_info>
  </SALES_ORDER>
</olin>
```

XPATH Document Shredding

XPath (or SQLX or SQL/XML) expressions are used to shred XML Documents. Shredding an XML document

enables you to store data elements into a relational database.

XMLSEQUENCE()

The XMLSequence() function returns a collection of XMLType. This function in a Table clause can be used to decompose the collection values into multiple rows. This can be further processed in a standard SQL query. The format clause ('/R/A[2]/B') indicates which child to return while the [2] designation breaks it down further signifying the node to return. The "extract('//B/text()')" clause in the select statement requests the data contents the of "B" node.

```
SELECT
value(T).extract('//Child/text()').getstr
ingval() XMLSequence_Example
FROM TABLE(XMLSequence(extract(XMLType('
<Root>
   <Parent>
        <Child>V1</Child>
        <Child>V2</Child>
        <Child>V3</Child>
   </Parent>
   <Parent>
        <Child>V4</Child>
        <Child>V5</Child>
        <Child>V6</Child>
   </Parent>
</Root>
'),  '/Root/Parent[2]/Child'))) T

XMLSequence_EXAMPLE
-----------------------------------------
V4
V5
V6
```

XPath Shredding Example – Putting It All Together

In our final example we will shred parts of an XML document containing parent nodes and children nodes. We will demonstrate the method by which we shred multiple child nodes and store them in a relational table.

```
<SHIPPINGRESULTS>
       …
  <CONTAINER>
    <CARTON>1</CARTON>
    <MSN>2710</MSN>

<TRACKING>1Z4R29X60200002014</TRACKING>
    <CHARGES>
      <TOTAL>36.76</TOTAL>
      <BASE>86.50</BASE>
      <ACCESSORIAL>
        <NAME>FUEL_SURCHARGE</NAME>
        <CHARGE>4.63</CHARGE>
      </ACCESSORIAL>
      <DISCOUNT>49.74</DISCOUNT>
    </CHARGES>
    <ORIGNINALCHARGES>
      <TOTAL>36.76</TOTAL>
      <BASE>86.50</BASE>
      <ACCESSORIAL>
        <NAME>FUEL_SURCHARGE</NAME>
        <CHARGE>4.63</CHARGE>
      </ACCESSORIAL>
      <DISCOUNT>49.74</DISCOUNT>
    </ORIGNINALCHARGES>
  </CONTAINER>
  <CONTAINER>
    <CARTON>2</CARTON>
    <MSN>2711</MSN>
    < TRACKING >1Z4R29X60200002023</
TRACKING >
        <CHARGES>
      <TOTAL>36.76</TOTAL>
      <BASE>86.50</BASE>
      <ACCESSORIAL>
        <NAME>FUEL_SURCHARGE</NAME>
        <CHARGE>4.63</CHARGE>
      </ACCESSORIAL>
      <DISCOUNT>49.74</DISCOUNT>
    </CHARGES>
    <ORIGNINALCHARGES>
      <TOTAL>36.76</TOTAL>
      <BASE>86.50</BASE>
      <ACCESSORIAL>
        <NAME>FUEL_SURCHARGE</NAME>
        <CHARGE>4.63</CHARGE>
      </ACCESSORIAL>
      <DISCOUNT>49.74</DISCOUNT>
    </ORIGNINALCHARGES>
  </CONTAINER>
  <SHIPMENTINFO>
    <ORIGINALSHIPVIA>

<CARRIERSERVICE>CAL.EXPRESS</CARRIERSERVI
CE>
    </ORIGINALSHIPVIA>
    <SHIPVIA>

<CARRIERSERVICE>CAL.EXPRESS</CARRIERSERVI
CE>
    </SHIPVIA>
  </SHIPMENTINFO>
</SHIPPINGRESULTS>
```

Figure V – Shipping Results XML Document

Step 1) Create a Cursor for Data Shredding

```
-- Cursors for Parsing SHIPRESPONSE XML
    CURSOR packages_cur IS
      SELECT EXTRACT(value(CONTAINER),
'//CARTONNUMBER/text()').getStringVal()
AS CARTONNUMBER ,
             EXTRACT(value(CONTAINER),
'//MSN/text()').getStringVal() AS MSN,
             EXTRACT(value(CONTAINER),
'//TRACKING/text()').getStringVal() AS
TRACKING,
             EXTRACT(value(CONTAINER),
'//LABELZPL/text()').getClobVal() AS
LABELZPL,
             EXTRACT(value(CONTAINER),
'//CHARGES/TOTAL/text()').getStringVal()
AS TOTAL,
             EXTRACT(value(CONTAINER),
'//CHARGES/BASE/text()').getStringVal()
AS BASE,
             EXTRACT(value(CONTAINER),
'//CHARGES/DISCOUNT/text()').getStringVal
() AS DISCOUNT
      FROM
TABLE(XMLSequence(EXTRACT(XMLType(lcl_res
ponse_message), '//CONTAINER')))
CONTAINER;
```

Note: lcl_response_message is an XMLType that contains an XML document similar to the one in Figure V.

Step 2) Write the SQL Query

```
FOR packages_row IN packages_cur
LOOP
     INSERT INTO TMS_SHIP_CARTONS_GT (
          SO_NUM,
          SO_REL_NUM,
          PRNT_DATE,
          CRTN_NUM,
          MSTR_SEQ_NUM,
          TRACKING_NUM,
          TOTAL_CHARGE,
          BASE_CHARGE,
          LABEL_ZPL )
     VALUES                    (
          soh_orders_rec.so_num,
          soh_orders_rec.so_rel_num,
          soh_orders_rec.prnt_date,
          packages_row.CARTON,  ←-----
packages_row:  data access through cursor
          packages_row.MSN,
          packages_row.TRACKING,
          packages_row.TOTAL,
          packages_row.BASE,
          lcl_label_zpl_char );
END LOOP;
```

Conclusion

XML development began on this project several months ago. We recently completed the majority of the SQLX (XML document creation) and XPath (document shredding) programming and began testing. Except for adjusting the SQLX queries and changes to XPath (parsing out similar tag fields under one parent), we made no major revisions. As far as the project is concerned, Oracle's XML DB is a stable environment and meets all the standards according to W3C. However, programming in XML DB does require a bit of some effort. The constructor XMLType creates an instance of an XML object. We used it to convert a CLOB into a properly qualified XML document for storage into a table with an XMLType column. When we shred XML documents we made extensive use of the function EXTRACT (with methods getstringval and getnumval). Once we developed the primitives for XML fragment creation and XML document shredding, we found ourselves using the same type of code over and over again. The iterative process for developing a project using Oracle's XML DB is quite similar and an extension to PL/SQL programming. We hope the information presented here establishes a good starting point for those embarking on a project using XML.

Biography

Coleman Leviter is employed as an IT Systems Software Engineer at Arrow Electronics. He has presented at IOUG's Collaborate 07. He is the WEB SIG chair and sits on the steering committee at the NY Oracle Users' Group. He has worked in the financial services industry and the aerospace industry where he developed Navigation, Flight Control and Reconnaissance software for the F-14D Tomcat at Grumman Aerospace. Coleman has a BSEE from Rochester Institute of Technology, an MBA from C.W. Post and an MSCS from New York Institute of Technology. He can be contacted at cleviter@ieee.org