

A Theoretical Paper using XML as the Integration Document for SAP Integration

November 4, 1999
Shaun Snapp

Glossary

- I. Introduction**
- II. A Brief Description of the Integration Points**
- III. XML Integration Architecture Diagram**
- IV. Components Required for an XML solution**
- V. XML Files & DTD for Integration**
- VI. XML Parser (The receiver of the XML document)**
- VII. XML Processor**
- VIII. The Processing Instruction Tag + its Relevance to Integration Workflow Management**
- IX. Conclusion**
- X. Appendix A: Viewing XML in IE5**

I. Introduction

The purpose of this document is to explain how XML may be used to facilitate i2-SAP application integration. The actual approach used at Company A was based upon IDOCS + Rhythmlink + Awk transformation scripts. Although this worked, the dynamic nature of input files rendered the solution somewhat labor intensive. That is changes in either i2 models (due to changing requirements) or changes in IDOCS (because of software upgrades or product hierarchy alterations) needed to be manually updated to the IDOC-Awk-Rhythmlink integration harness.¹ The purpose of this exploratory paper is to examine whether XML should be used to perpetuate SAP or i2 field level changes into an integration harness in an automated fashion.

The paper is not designed as an XML primer. There are many good on the subject (many are footnotes in this paper). For the uninitiated a list of XML's properties is listed below:

XML is...

- A descendent of SGML (Standard Generalize Markup Language)
- A document standard concerned with the structure of data
- Multi-purpose. Application integration is just one of its many uses. In general XML is useful for adding structure to data.
- A markup language, not a procedural language
- Designed primarily for machine to machine communication

XML is not...

- Primarily concerned with the display of information. An XML document may be configured in display by an XSL (Extensible Style Language, a subsidiary and complementary standard) However, as XML is designed for machine communication, the ability to view XML in a browser is not its reason for being, particularly for our purposes of application integration.
- A sub-unit of HTML
- (does not) Describe a physical communication method. The FTP-ing of an XML file between two companies is still EDI. While EDI describes both a document format, a form of communication, XML only describes the structure of data.

¹ A detailed explanation of the IDOC/LIS methodology used to integrate with Company A's SAP system is described in The Company A Integration Case Study which currently resides out on i2 Lotus Notes in the SAP integration database.

II. A Brief Description of the Integration Points

After having implemented the Company A solution, with i2 performing all IDOC translation, one basic issue became apparent during the project and into testing. The maintenance of a changing hardcoded solution based changing IDOCS and/or changing i2 models was onerous. This manual workload performed by i2 was one of the primary motivations for looking for another intermediate document method. (ROI is a SQL solution, hence we needed another technology in order to complete an intermediate document analysis.)

In order to keep this paper simple we have taken a subset of the Company A solution around which to demonstrate our integration methodology. Figure 1 describes the interfaces to be written. All five are inbound interfaces providing data to either DP or SCP in order to support a buy process (Company A's purchase of materials from their factories). The IDOC creation time column indicates how long it took the SAP system to create a single export of the file we had requested. Previous to using an XML solution we were required to transform the SAP IDOC into a file format acceptable by Rhythmlink. With the XML solution we will rely upon the XML Translator described in the following section.

Figure 1. (Objects Removed as they are client specific)

III. Architecture for XML solution

In Figure 2 we have a diagrammed view of the how the components described above would come together to form our solution. Notice we transform the format of the data 4 times prior to loading into the DP database (IDOC → XML → CDM → DP Flat File) As SCP is fed from ADW tables, we transform its upload file only 3 times before it is SCP readable. SQL Loader is used in addition to Rhythmlink. SQL Loader gives us a little more speed as we can disable the referential integrity.

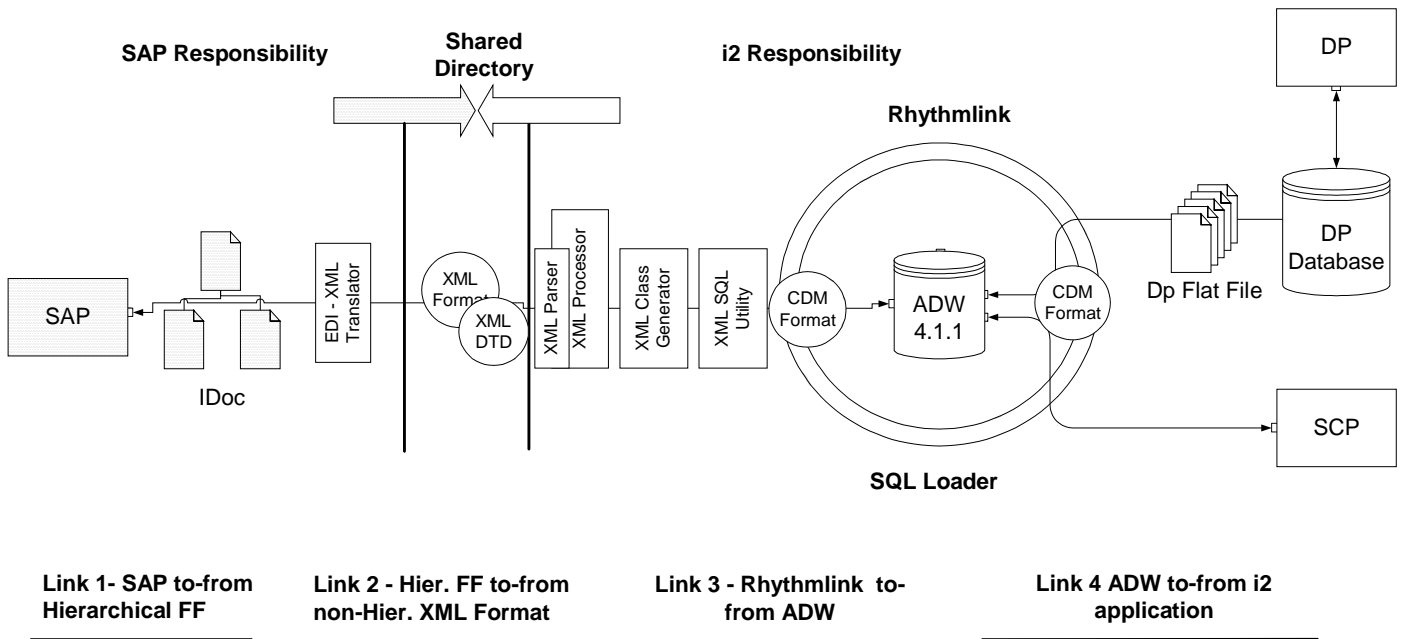


Figure 2.

IV. Components Required for an XML solution

In order to implement our solution using XML we would need the following components². We will address each lightly here. Later in the paper we will delve into the details behind the XML

² Being a new technology it is relatively difficult to find published shortcomings of XML. Here are some of the ones noted by Rick Strahl in [Taking Advantage of XML for Messaging](#):

- XML conversion is very resource intensive (both in terms of CPU resource to create and unpack XML into native data as well as the actual data size)
- The current XML parsers are fairly slow

XML documents can bloat data considerably with all of the markup tags. This is somewhat axiomatic as XML is one of the few human readable document formats. When sending data over the wire this can be a problem if data is not compressed (of course this is minimal compared to the bloat which accompanies an IDOC. Regarding this, while actual used data is an IDOC is roughly 15% in the Company A IDOCs, XML has a usable data percentage of generally 40% - depending on the XML object) I think you are referring to the relative verbosity of the XML code. Both IDOCs and XML objects are verbose. IDOCs are verbose because they are essentially complete business objects whose field compositions are determined by SAP. SAP has a transaction processing/mainframe view of the world. Very structured, very reliable, but very inflexible. Also, as IDOCs are not extensible, they must pass the entire universe of fields

Parser, Processor + EDI Translator. The Class Generator and XML SQL Utility will not be further elucidated as they are rather standard utilities:

- EDI – XML Translator (a XML SQL Utility may not be used with SAP unless ROI is used data must be exported and imported to/from SAP with the nefarious IDOC.)
- XML Parser – Parse: literally to determine the structure of an object. The Parser is a component of the XML Processor. It is used for interrogating the XML object and verifying validity (conformance) against the DTD file. Parsers maintain a built-in Error Recovery until fatal error.³ They enforce the structure of the document before sending the structure on to the SQL Utility. An invalid XML document will have a difficult time being loaded into the database schema.
- XML Processor – Used to read XML documents and provide access to their content and structure. It is assumed that the XML processor is doing its work on behalf of another module, called the application.
- XML Class Generator for Java⁴ - Creates Java source files from an XML DTD. This is useful when an application wants to send an XML message to another application based on an agreed-upon DTD. Using these classes, Java applications can construct, validate, and print XML documents that comply with the input DTD.⁵

(Actually, this Class Generator appears to be more a programming productivity tool that necessity. This would allow a programmer to create new instances of the XML classes from a class library built up over time from sending and receiving XML documents)

- XML SQL Utility for Java – For import and export of validated XML fields into Oracle ADW table. Generates an XML Document from SQL queries, writes XML data to a database table.⁶

V. XML Files & DTD for Integration

Now that we have the basic components defined, we will turn our attention to the actual integration objects, and describe the XML files that will be received by our XML Parser. XML files are extensible and therefore flexible. It allows us to define whatever field combinations we desire. After we have the XML Object defined, we configure our DTD so that our XML Parser understands our file structure. This is a dramatic departure from the use of any static data file (ie. IDOC or generic flat file). Secondly, this distinction is essentially the essence of XML.

The Difference Between Changing IDOCs and Changing XML

The difference is that with XML, field additions or subtractions, are communicated both in the data file (as in an IDOC) and in the document definition (unlike an IDOC). Therefore as long as the parser is programmed with the full universe of possible fields, XML documents may be

associated with a business transaction (a la EDI). XML is verbose because it is pedantic, i.e., it explains its contents in specific detail. This being the price of being human readable. So, the cost is transferring more bytes across the wire. The benefit is ease of use and improved troubleshooting.

³ XML: The Annotated Specification – Bob DuCharme

⁴ XML in Oracle8i – Martin Boyd – XMLFocus (Periodical) November 1999

⁵ Oracle website - <http://technet.oracle.com/tech/xml/classgen/>

⁶ You will note a bias in the components toward Java. The reasoning for this is that Oracle 8i has XML compatibility. Oracle recommends the use of Java based XML tools for integration with Oracle 8i. At the time of this paper, Company A ran Oracle 8. Happily for us this is a theoretical paper, as such we get to choose our “optimal” database platform.

dynamically changed, without leading to manual re-mapping work or human intervention of any kind. IDOCs can never perform in this way because they have no definition to accompany them. Secondly integration to an IDOC is hard-coded. One field moves, and the transformation script must be changed.

The following files would apply for the proposed integration. These are five SAP outbound documents. For brevity I will only include one item (i.e. sales order, PO) per file example.⁷ There are two basic XML classes: the XML object, and the DTD. The XML objects contain both the character data to be passed to our receiving application + the markup which defines the character data. The DTD is the Meta data definition of the XML. The objects below were validated at Brown University's Scholarly Technology Group (STG) home page - <http://www.stg.brown.edu/service/xmlvalid/>⁸

Figure 3.

<p>1. A – SalesOrders</p> <pre><?XML VERSION="1.0" RMD="NONE"?> <!DOCTYPE SALESORDERS SYSTEM "SALESORDERS.dtd"> <SALESORDERS> <SALESORDER> <COUNTRY>CANA</COUNTRY> <SCG>S1</SCG> <CHANNEL>001</CHANNEL> <STYLECOLOR>10205214100</STYLECOLOR> <SEASON>FA99</SEASON> <QUANTITY>23000</QUANTITY> <GLOBAL>1</GLOBAL> <REGION>AM</REGION> <SUBREGION>AM</SUBREGION> <OWNERSHIP>N</OWNERSHIP> <ALLCOMPANY A>1</ALLCOMPANY A> <BUSINESSUNIT>000</BUSINESSUNIT> <LINE>02</LINE> <STYLEGROUP>02</STYLEGROUP> <MAXPRICE>014782</MAXPRICE> <TYPEGROUP>\$0000</TYPEGROUP> <PRODUCTTYPE>01</PRODUCTTYPE> <DATE>07202000</DATE> <VAS>01</VAS> <FILETYPE>Z003</FILETYPE></pre>	<p>1. salesorders.dtd</p> <pre><?xml version = "1.0"?>⁹ <!--salesorders.dtd --> <? SCP Workflow ="Consumer Package Goods for SCP.dat" ?> <!ELEMENT aggregatesalesorder (country,scg,channel,stylecolor,season,quantity,glob al,region,subregion,ownership,allCompany A,businessunit,line,stylegroup,maxprice,typegroup,p roducttype,date,vas,filetype+)¹⁰, <!ELEMENT country #PCDATA> <!ELEMENT scg #PCDATA> <!ELEMENT stylecolor #PCDATA> <!ELEMENT season #PCDATA> <!ELEMENT quantity #PCDATA> <!ELEMENT global #PCDATA> <!ELEMENT region #PCDATA> <!ELEMENT subregion #PCDATA> <!ELEMENT ownership #PCDATA> <!ELEMENT allCompany A #PCDATA> <!ELEMENT businessunit #PCDATA> <!ELEMENT stylegroup #PCDATA> <!ELEMENT maxprice #PCDATA> <!ELEMENT typegroup #PCDATA> <!ELEMENT producttype #PCDATA> <!ELEMENT date #PCDATA></pre>
--	--

⁷ Notice the appealing feature of XML being human readable in its native format. The integration files, which we used, for the actual Company A integration were indecipherable without the data definition.

⁸ A surprising fact according to Brown STG is that the majority of XML out on the Internet is not valid. While invalid XML can still be of use, for application integration invalid XML documents bring us back to zero, without validity we can not have machine-to-machine interoperability. The author leaned the strickness of the XML spec when validating the documents above at the Brown STG site. It took some time, but the objects above are fully valid.

⁹ There currently is no XML version > than 1.0. Declaring the version is more about simplifying the parsing routines for when there is a new XML version.

¹⁰ This is a section declaration. It indicates what elements are to follow, and whether they are optional (?), may be zero (*), or may have multiple values (+). In this case each element should appear only once, and each is necessary. This is the default with the section declaration, so we have no need to use one of the three symbols.(*,?,+)

<pre></SALESORDER> </SALESORDERS></pre>	<pre><!ELEMENT vas #PCDATA> <!ELEMENT filetype #PCDATA >¹¹ ></pre>
<p>2. D – Country Inventory</p> <pre><?XML VERSION = "1.0"?> <! DOCTYPE COUNTRYINVENTORY SYSTEM "COUNTRYINVENTORY.DTD"> <COUNTRYINVENTORY> <COUNTRY> <ITEMNAME>136003141</ITEMNAME> <COUNTRY>US</COUNTRY> <QUANTITY>1000</QUANTITY> </COUNTRY> </COUNTRYINVENTORY></pre>	<p>2. countryinventory.dtd</p> <pre><?xml version = "1.0"?> <!--countryinventory.dtd --> <? SCP Workflow ="Consumer Package Goods for SCP.dat" ?> <!ELEMENT countryinventory (itemname, country,quantity) <!ELEMENT itemame #PCDATA> <!ELEMENT country #PCDATA> <!ELEMENT quantity #PCDATA></pre>
<p>3. E – Purchase Orders</p> <pre><?XML VERSION = "1.0"?> <! DOCTYPE PURCHASEORDERS SYSTEM "PURCHASEORDERS.DTD"> <POS> <PO> <ITEMNAME>136003141</ITEMNAME> <COUNTRY>US</COUNTRY> <SOURCEFACTORY>US</SOURCEFACTORY> <QUANTITY>1000</QUANTITY> <SHIPDATE>1000</SHIPDATE> <NEEDDATE>1000</NEEDDATE> <POID>US</POID> </PO> </POS></pre>	<p>3. purchaseorders.dtd</p> <pre><?xml version = "1.0"?> <!--purchaseorders.dtd --> <? SCP Workflow ="Consumer Package Goods for SCP.dat" ?> <!ELEMENT purchaseorders (itemname, country, sourcefactory,quantity, shipdate,needdate, poid) <!ELEMENT itemame #PCDATA> <!ELEMENT country #PCDATA> <!ELEMENT sourcefactory #PCDATA> <!ELEMENT quantity #PCDATA> <!ELEMENT shipdate #PCDATA> <!ELEMENT needdate #PCDATA> <!ELEMENT POID #PCDATA></pre>
<p>4. G – Global Delta</p> <pre><?XML VERSION = "1.0"?> <!DOCTYPE GLOBALDELTA SYSTEM "GLOBALDELTA.DTD"></pre>	<p>4. globaldelta.dtd</p> <pre><?xml version = "1.0"?> <!--globaldelta.dtd --> <? SCP Workflow ="Consumer Package Goods for</pre>

¹¹ XML & DTD Component Explanation for the Above Files

ELEMENT -- As you can see from the above, the one of the basic components of the DTD is the ELEMENT tag. This is the "field" we intend to pass or be passed, i.e. they are analogous to the rows in relational tables. An element may contain listings, figures, examples or tables. In our case simply defining it a single field will suit our purposes. Following the ELEMENT tag is the field name also referred to as the generic identifier.

ENTITY – for XML an ENTITY is a storage element. This can be used to either hardcode an ELEMENT to a specific value which is referred to as an internal entity (i.e. <ENTITY pubyear "1999">) , or declare an file to be used with XML (i.e. <ENTITY flowerpick SYTEM "flower.jpg" NDATA JPEG>) Actually, both an XML document and DTD are also ENTITY(s) by the receiving system.

#PCDATA— #PCDATA refers to parsed character data, otherwise known as XML text. According to the XML specification, this can be just about any combination of text-based characters and white space. PCDATA and CDATA escape parsing, and are understood by the parser as character data.

ATTRIBUTES – Information about ELEMENTS. They allow you to constrain the possible value of ELEMENTS. No ATTRIBUTE was used in the above DTD, however if we wanted to constrain ELEMENT values we could. Our ATTRIBUTE type would be a string type for normal character data. The opposite of what we did in this DTD is to use an Enumerated ATTRIBUTE type. This is where we would declare approved values. This is accomplished with the NMTOKEN tag i.e.

```
<!ATTLIST salesorder country NMTOKENS #IMPLIED><country="US CA BE JP"></salesorder>
```

Source of the above www.WebTechniques.com – The Death of DTD - Michael Floyd

<pre> <GLOBALDELTAS> <GLOBALDELTA> <ITEMNAME>136003141</ITEMNAME> <SOURCEFACTORY>FT</SOURCEFACTORY> <DELTAQUANTITY>-274</DELTAQUANTITY> <FACTSTARTDATE>07201999</FACTSTARTDATE> </GLOBALDELTA> </GLOBALDELTAS> </pre>	<pre> SCP.dat" ?> <!ELEMENT globaldelta (itemname, sourcefactory, deltaquantity, factstartdate,) <!ELEMENT itemame #PCDATA> <!ELEMENT sourcefactory #PCDATA> <!ELEMENT deltaquantity #PCDATA> <!ELEMENT factstartdate #PCDATA> </pre>
<p>5. H – Global Standard Size Curve</p> <pre> <?XML VERSION = "1.0"?> <!DOCTYPE GLOBALSTANDARDSIZECURVE SYSTEM "GLOBALSTANDARDSIZECURVE.DTD"> <GLOBALSTDSIZECURVES> <GLOBALSTDSIZECURVE> <STYLECOLOR>136003141</STYLECOLOR> <SIZE>8.5</SIZE> <PERCENTAGE>12.5</PERCENTAGE> </GLOBALSTDSIZECURVE> </GLOBALSTDSIZECURVES> </pre>	<p>5. globalstandardsizecurve.dtd</p> <pre> <?xml version = "1.0"?> <!--globalstandardsizecurve.dtd --> <? SCP Workflow ="Consumer Package Goods for SCP.dat" ?> <!ELEMENT globaldelta (stylecolor, size, percentage) <!ELEMENT stylecolor #PCDATA> <!ELEMENT size #PCDATA> <!ELEMENT percentage #PCDATA> </pre>

VI. XML Parser (The receiver of the XML document)

The XML Parser is ½ of our application API.¹² For this case study we are implementing a validating parser. Validating parsers are necessary for our purposes as we must be exact in matching the XML file to its DTD. This will also move us in the direction of automating the movement of the business objects between our applications. If our document/object is shown as invalid, the parser will generate errors and only a portion, or none of the XML document may be loaded into our relational schema. After validating, the second and final function of our parser is to decompose the XML document instance.¹³

We used the Microsoft XML Parser in order to parse the documents for integration points A, D, E, G and H. The Microsoft XML Parser runs from an instance of IE5 or from a command line in DOS. The UI is below as Figure 5. Notice that we have parsed integration object A: Aggregate Sales Orders. We can be assured that this document is well formed. If it were not, the error box at the bottom of the UI would display that the document had not been properly parsed.

The MS XML Parser run from the command line is how we would proceed for the actual integration. The UI below is mostly for the designing and verification phase. The parser works off of the `<!DOCTYPE SALESORDERS SYSTEM "SALESORDERS.dtd">` tag in the XML document in order to validate against the corresponding xxx.dtd file.

Figure 5.¹⁴



¹² Designing XML Internet Applications – Michael Leventhal

¹³ The parser used to facilitate this project was the Microsoft XML Parser Redistributable downloaded off the Microsoft site for free.

VII. XML Processor (Mapper to Schema and Initalizer of SQL Utility)

The other ½ of our API, or at least the logic for the API is the XML Processor. Once we have a firm understanding of the business object structure, we can progress to the mapping of the extracted fields into our database schema (in our case the ADW). In an identical fashion as we traditionally use Rhythmlink, the logic below is instructive as to the type of logic which must be programmed into the XML Processor. It is also the actual logic used to move integration object E: Open Purchase Orders into our APPR_SHPMNT ADW 4.1.1 table.

APPR_SHPMNT (Directly populates the ADW)

RhythmLink Process	Source Catalog: And Source Table(s)	Destination Catalog: And Destination Table(s)	Comments
JOIN_TABLE	<i>FF_Catalog</i> SAP_Purchase_Orders & <i>ADW_Catalog</i> PLN	<i>Integration_Point_E</i> JOIN_Purchase_Orders_PLN	No join predicate. All possible permutations are created. The input file SAP_Purchase_Orders is located at <i>/i2dev/sandbox/4.1.1/rl/data/work/SAP_Purchase_Orders.dat</i>
COPY_MAP Map_SAP_ADW_ E_APPR_SHPMNT	<i>Integration_Point_E</i> JOIN_Purchase_Orders_PLN	<i>ADW_Catalog</i> APPR_SHPMNT (CLEAR_BEFORE_ADD)	

Figure 6.

VIII. The Processing Instruction Tag + its Relevance to Integration Workflow Management

Certainly, the architecture above could work. And with less manual intervention than we currently are required to do on our current solution. However, the XML solution above is only an incrementally better solution than the undefined IDOCs we currently use. What is missing is re-usability and concept of workflow or methods.

A method is essentially a term borrowed from OO programming.¹⁵ OOP depends upon the concept of abstract entities called software objects. These range from items on a screen such as text boxes to stacks and queues. Every object in OO has both a state and a method. For our purposes, an XML document can be our object, and the number 900 in a quantity field would be the state of one particular field. Therefore, the XML object would be made up of a series of fields with various states, with the overall combination of fields providing a state to the complete XML document. Basically a method determines what an object is capable of doing with a series of

¹⁴ For those of you planning to use the Microsoft XML Parser, you should know that it has a strange feature. Documents may only be brought into the UI if stored in the C:/msxml/viewer directory. Perfectly good XML documents stored anywhere else will not be read by the parser.

¹⁵ Client-Server Information Systems - James Goldman

other objects. For our purposes of application integration a method would determine what to do with say an XML document called Open Purchase Orders (i.e. what tables to populate in our ADW, when to populate, etc...)

Now with that background we can move to analyzing an XML document and comparing it to a request for a method included within.

So far we have not addressed methods. The XML spec contains something called a Processing Instruction (PI). A PI allows documents to contain instructions for destination applications. If you look at the DTD in figure 3. You will notice the PI in the declaration. (i.e. `<? SCP Workflow ="Consumer Package Goods for SCP.dat" ?>`)

In order to interpret the `<Consumer Package Goods for SCP>` tag there are two options we could follow for the parser:

Localized Solution

1. Buy a standard XML parser and insert the logic and mapping to the ADW differently per implementation. The problem with this is it is manual, and low in re-use

Productized Solution

2. Create an i2 Parsing Product (maybe: Rhythm Optimal X-Parser (ROXP)) which could read workflow tags:

```
<? SCP Workflow ="Consumer Package Goods for SCP.dat" ?>
<? SCP Workflow ="High Tech for SCP.dat" ?>
<? DP Workflow ="Metals for DP.dat" ?>
<? DP Workflow ="Baby Food for DP.dat" ?>
etc....
```

The ROXP would read `<Consumer Package Goods for SCP>`, and the object definition tag of `<business object><Country Inventory></business object>` and map the data fields as per the programming. The Processor would then invoke the SQL utility to load the Receipt and On_Hand_Lvl tables in the ADW. This product would need to be released in concert with new ADW releases.

Finally, SAP would have to agree to send the appropriate "method" tags in their XML documents. Additionally, with an Internet connection we could pass messages between any of our planning engines with this architecture (either between companies, or between instances of i2 engines over a corporate network).

IX. Conclusion

Clearly, substitution of a structured XML document maintains significant benefits over a flat file or IDOC structure flat file approach. The advantages are as follows:

- Dynamic capability to send new documents (business objects) with the XML parser performing the work currently being performed manually
- Better, clearer definition of the business objects which are intelligible to both human and machine alike
- Ability to pass business objects between i2 applications through the same methodology. Ability to pass objects between i2 applications over an internet either intra or extra enterprize

The XML solution is not without its costs and they are enumerated below:

For The Localized Solution

- Purchase of XML components
- Training on XML components
- Programming of XML Processor (i.e. mappings between ELEMENTS and tables in the ADW, switching logic triggered by Processing Instruction tags) – Essentially replacing the Rhythmlink coding.

I2 Productized Solution

- For full benefit, development of an i2 XML Processing product with both i2 to SAP mappings + PI logic (released in lock-step with new ADW releases)
- Convincing ERP vendors to invest in IDOC (and Oracle/Peoplesoft/Baan intermediate document) translator. SAP has gone on record as supporting IDOC-XML translation. However, SAP has a tendency of announcing a lot of things, which never actually happen. Third party packages such as Mercator do provide IDOC-XML translation software.

One final caveat. At Company A we have not implemented an XML solution. After our initial investment in a flat file methodology it would be difficult to invest in an alternate solution. However, performing the integration the “old fashioned way” has led us naturally to some conclusions regarding how we would implement such an integration given a clean slate. This paper is a theoretical exercise. The effort was made to make it a well-researched theoretical exercise, but as implementers it is our opinion that nothing replaces the actual experience.¹⁶ⁱ

¹⁶ Special thanks to the peer reviewers of this paper. The following people contributed in either working with the author through some contributory thought process and/or providing assistance through various sticking points (code generation problems, object-oriented discussions, etc.) They are as follows:
Gurdip Singh
Satish Sandhir

V. Appendix A: Viewing XML in IE5

Although HTML and XML are two different markup languages with two different purposes, their common ancestry of SGML is of benefit in the following example. All that needs to be done in order view an XML document in a browser is change its extension to xxxx.htm. The changed document will automatically open into IE5 from the Windows Explorer. Figure 7 is a picture of how IE5 displayed the XML document Sales Order Object. Notice how all markup tags have been stripped out leaving only the original parsed character data. IE5 has parsed the object without error. From this we can surmise that HTML and XML are compatible. Figure 8 shows the screen when we right mouse-click the IE5 window and select a view of the source code. All the tags and XML is still there.

Figure 7.



Figure 8.

