

IDEA Moving & Storage™: ISO-based Document Management

Ron Turner, Soph-Ware Associates, Spokane WA

Introduction:

Three years ago the DOE (OSTI in particular) issued a solicitation through the SBA's Small Business Innovation Research (SBIR) program for an architecture to support distributed multimedia. The main problem with distributed heterogeneous data objects is that we cannot be certain at any moment where those objects are or even what they are. Soph-Ware Associates' proposal was to design and implement a secure client-server architecture to publish distributed multimedia. Like the data, the software can be heterogeneous, allowing the systems designer to pick and choose among the most appropriate COTS products.

Soph-Ware is currently piloting and will soon deliver version 1 of a working implementation of that architecture, known as Intelligent Document Exchange Architecture (IDEA). Under terms of the SBIR, Soph-Ware seeks to commercialize its funded R&D as a product. That product is called IDEA Moving & Storage. It comprises the document management activity of IDEA and is the subject of our discussion at this meeting.

IDEA Architecture

Figure 1 is a schematic data flow diagram depicting the various services that are part of IDEA. The approach of IDEA is for the software to function as the middle tier in a three-tier architecture separating data objects from the various processes that manipulate those objects. The Patron in this diagram is separated from the document store, and the various services are likewise separated from one another. The diagram also demonstrates that IDEA is itself an embedded client/server system with its own server and the various services as the server's clients.

Figure 1 depicts a generic document retrieval session. The Patron, having already searched the document store, issues a request to the SGML Information Server. In this secure architecture, the Client Registry must certify the patron. In addition, the Security Manager must validate his or her access to each of the objects comprising the requested document. If this transaction is fee-based, the Business Manager must verify the patron's subscription or debit the patron's account. The Entity Registry and File Manager actually retrieve the document objects. The Workstation Registry, because it knows the profile of the patron's current workstation, supplies the Transformation Manager with sufficient information to prepare the document for final delivery.

SGML for Document Management

Soph-Ware Associates specializes in ISO 8879 Standard Generalized Markup Language (SGML). And IDEA is especially useful for manipulating documents that are stored as SGML. However, the most important aspect of IDEA is not *what* kind of data objects it handles but *how* it handles those objects. IDEA exploits a powerful built-in capability of SGML for document

management. IDEA is therefore an SGML architecture first because it incorporates SGML document management. The documents and media it manages may be SGML or any proprietary or legacy format. At a lower level, The interservice messaging in IDEA consists of SGML packets. Furthermore, the metadata (bibliographic header information) for each document object is stored as SGML documents. So whether the document store itself is SGML or not, IDEA exploits the standard in every way possible. Just as SGML documents continue to offer the best hope for their longevity, this SGML architecture promises to persist, regardless of inevitable changes in software, hardware architectures, operating systems, platforms, scalability, and speed.

The thick-lined loop in Figure 1 encircles the document management activity of IDEA. Soph-Ware is now packaging that portion as IDEA as Moving & Storage, a middleware product to support document control for large SGML installations.

Moving & Storage: the SGML Connection

Those who are familiar with SGML (or XML, its kinder, gentler offspring) no doubt think first of pointy brackets for tag names, the same apparatus used for HTML on text for Web pages. But SGML is about much more than markup for text. SGML markup is for describing relationships among *structural elements* of documents. But the standard also provides a methodology for describing the relationships between the storage entities in a document and their actual physical locations. That methodology is the principal feature of Moving & Storage.

The SGML standard, via its Extended Facilities, addresses the persistent locator problem by applying a technique that is common in programming: indirection. Rather than hard-coding a value in a program, it is better practice to use a variable. And it is even better practice to use a pointer to a variable. (This is more or less what a “handle” is.) The principle is, the greater the degree of indirection, the better. Conversely, the less coupling the better. It is better because indirection allows for a single data object to be shared by many processes. The maintenance payoff for data locator indirection is that by updating a single data object in one location, you are simultaneously revising five, 50, or perhaps 5000 documents or reports or pictures or whatever the data may be. The penalty for indirection is the front-loading cost of doing it right, because indirection adds initial overhead. Figure 2 contains a tiny example of how we apply SGML to achieve indirection for document objects.

Note first that we must understand the slightly subtle distinction between an *entity* and an *element*. An entity is a physical thing—a storage thing—defined formally as simply a collection of characters. The entity has to do with where and how it is to be found, and it is oblivious to its role(s) within the document(s). We define an element, on the other hand, solely by its relationship within the single document in which it occurs. That relationship is a relationship of structure. In our example the paragraph element, tagged as <para>, is probably a subelement of <section>, which is part of <chapter>, which is part of <unit>, which is part of <book>, which is part of <set>. An element therefore is a structural thing, a relational thing.

In our example an entity is embedded in an element. Note, by the way, that element-entity is not a part-whole relationship. The entity `yonder` is a name, like a variable in a computer program. This hints at the notion of indirection discussed earlier, and we are about to trace exactly how that works. From its name we don't know whether `yonder` refers to a single symbol like a copyright symbol or company logo, a dynamic string like date and time, a piece of dynamic data like this hour's Dow Jones Industrial Average, a JPEG graphic, a video, or an audio file. From reading the example we only know that it's something "out there," something that the SGML system needs to fetch and insert into the paragraph, replacing the string `&yonder;`.

Clearly the SGML system (whether it's an authoring tool or a publisher's retrieval product) needs certain information in order to fetch what `&yonder;` is referring to. Those pieces of information show us how an SGML document is "bolted together." The SGML parser recognizes first that `yonder` is an entity, something that needs replacing. It knows that because of the "&" and the ";" surrounding the name. It then expects to find that entity declared somewhere in the document itself (the "document instance"). That entity declaration points to yet another name `doc2`. The parameters of the entity declaration are significant:

1. The string `<!ENTITY yonder` informs the parser that this is the declaration it is looking for. (As a matter of procedure, this declaration, along with perhaps hundreds more for a complex document, may be in a separate physical file.)
2. `SYSTEM` informs the parser to use some system-based process to locate the entity. That process is defined by what follows.
3. `<IDEA>doc2` is a formal system identifier, another construct that is part of the SGML standard. This functions as the "storage object specification (SOS)." `<IDEA>` is the "storage manager name." "Storage manager" is a concept described at length in the SGML standard. The "entity manager" (also an SGML concept) manages the mapping of virtual objects (entities) to real objects (files). The details of how the entity manager maps a virtual storage object (here, the entity `yonder`) to an actual (physical) location are not mandated by the standard. But SGML does prescribe how the entity manager maps a virtual object to a storage manager.
4. `doc2` is the name within the `IDEA` storage manager name space. Within `IDEA` that name is a unique entity. Outside `IDEA` it has no significance. Entering that entity name plus its declaration into the system is an administrative activity, part of registering the document into `IDEA`'s Entity Registry. (The Entity Registry is not the same as the "entity manager" of the standard.)
5. The parameter `<IDEA>` designates the SGML storage manager. We have implemented this manager with the Entity Registry. It contains, among other things, its own file of entity declarations. It contains a declaration for `doc2`.

6. This declaration—the one for `doc2` in the Entity Registry—encapsulates its own storage method. The method is proprietary to IDEA. At this time IDEA uses FileNet’s document management product FileNET Document Services as its method. It can use any other document management product. Note that there is no mention or implication of the underlying data base product or data base architecture that supports the method.
7. In the example, the FileNet system’s entity name is `id2`, the name by which IDEA’s File Manager will locate and retrieve the actual stored element.

All of this apparently “excess SGML baggage” is the overhead we described above. This clearly requires careful software design. And what have we gained with this degree of indirection and layering of services? First, we could conceivably have not just one entity manager (IDEA) but several: IDEA2, IDEA3... Second, the architecture is not locked into a particular data base. The underlying data base could be Microsoft SQL Server, Sybase, Oracle, or whatever the document management system will support. Third, we are not locked into a particular vendor’s document management product. If we chose to substitute a product from Documentum, for example, instead of FileNet, we would need to alter only the catalog (Entity Registry). Fourth, and of greatest importance to the administrator, we only need to update or revise the actual data object in a single place.

Although the mapping from virtual document objects to actual stored data seems tortuous, Moving and Storage hides this from the writer, editor, or document technician. Within an authoring/editing tool (Adobe’s FrameMaker+SGML, for example) Moving & Storage prompts only for minimal input. The author only needs to know the name of the storage manager (IDEA) and the name of the entity within that storage manager (`doc2`). It appears to the user as simply a disk name and file name, for example. The Patron, on the other hand, who is operating all this only for retrieval sees nothing of the underlying structure.

The tiny example we studied shows an entity to be contained within an element. The more frequent case is for large clusters of text, comprising many elements, to be part of a single stored entity. For instance, a “master document” element, `<manual>` say, may contain nothing but entity references, each pointing to a chapter’s worth of text. But the method of mapping entities from their reference names to their actual locations is the same.

Moving & Storage strives to maintain its SGML integrity as an *entity* manager. That means that it fragments, stores, retrieves, and reconstitutes the pieces of a document as entities. Some other management systems use the *element* as the storage unit. While either approach calls for a database solution, and while either is amenable to document management, the consistent preservation of SGML entities assures maximum portability, standards compliance, and document longevity with the minimum consumption of resources. There are dramatically fewer files required for entities than there would be for elements.

Moving and Storage: the Database Connection

In IDEA every one of the services sits on and is driven by a data base (see the vertical bars in figure 1.) The data base for Moving & Storage consists of SGML entity declarations and of bibliographic header information (metadata), themselves stored as SGML documents. We have only casually stated that that data resides in a data base. It is important to note that SGML is indifferent to data structures. Each document object could just as well be a flat ASCII file.

But in the real world of large documents (multi-volume, multi-set), one cannot think of document management without considering data bases. There are three reasons why the systems designer must keep data base strategies clearly in mind. First, structured data most logically should reside in a data base, not because it's SGML data, but because that's the best way to store data for large systems—of anything, including documents. Second, those objects, properly structured and intelligently stored, can function as reusable, shareable data objects. Third, the information borne by richly marked SGML documents (via attributes and declarations) can constitute a high value added to the document store. If that information is properly structured (as in a data base), the documents can be mined as well as simply located. All such data warehouses or data marts require a rich interaction of data bases.

Moving & Storage: the Document Management Connection

Document management means several things. That's because it can serve several purposes. It can trace the progress of a document through the workflow of a collaborative writing group. It can do simple check-in/check-out control to assure that only a single copy is open for write access. It can provide a mechanism for controlled access to secured documents. It can help implement e-commerce with fee-based materials. Most of all, it is a standards-based answer to the quest for persistent locators. Moving & Storage uses document management to implement an SGML-prescribed method for entity location and control.

The big problem that exists for heterogeneous distributed document objects is the classic problem of locator persistence. This is IDEA's reason for existence. We choose to manage these objects, not only because it's trendy or because there are lots of them, but because it is the only way to maintain the integrity of a document entity's location. That managed object and the documents that "own" it remain totally intact, no matter how radically that object may be revised or relocated.

Figure 3 depicts the relationship between (1) stored document objects (SGML/XML entities or data in other formats), (2) an off-the-shelf document management system, enhanced to be SGML-compliant, (3) Moving & Storage, and (4) SGML/XML tools.

Figure 1

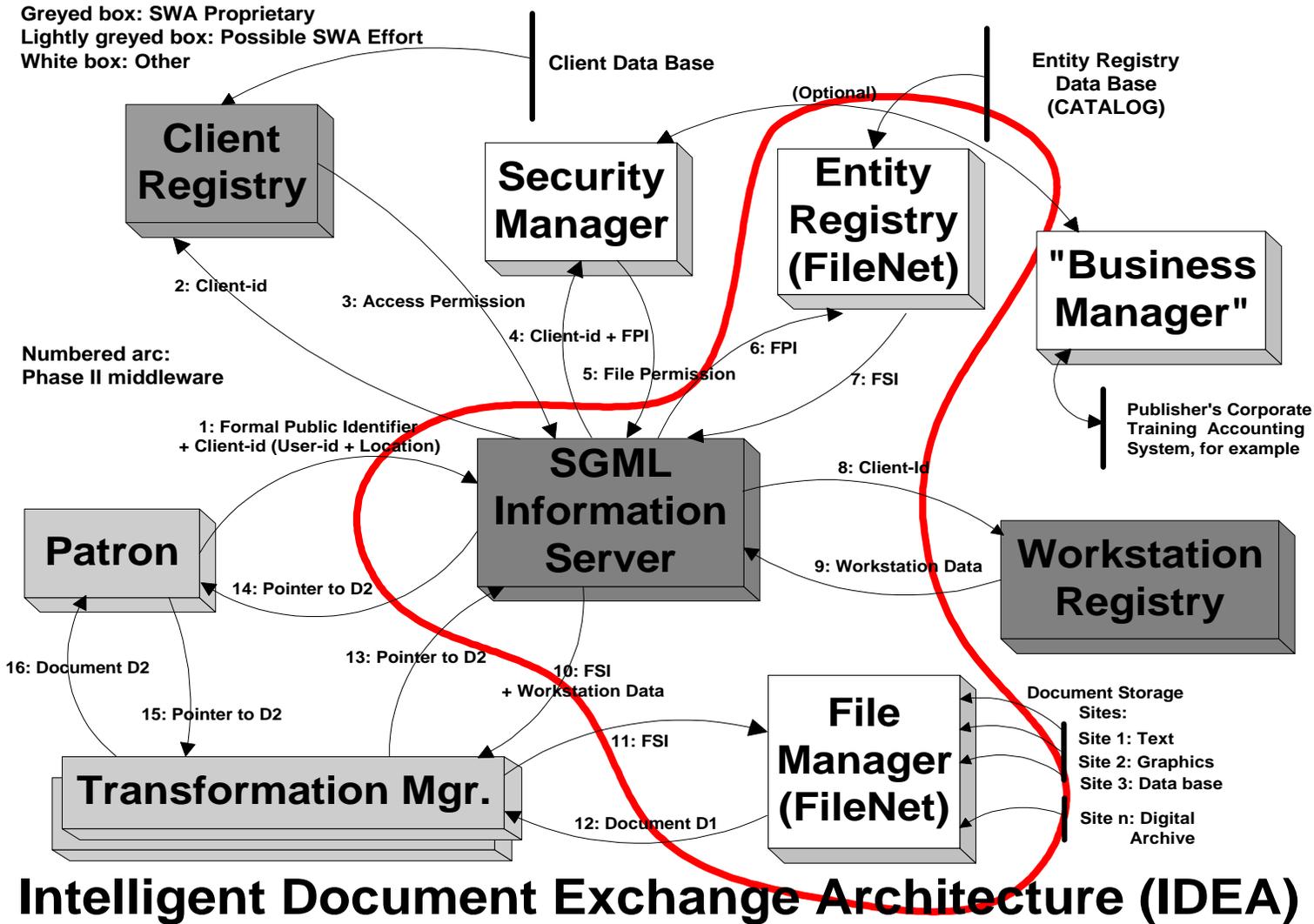


Figure 2

So What's New?

SGML Document Management μ -tutorial

■ Essential definitions

- ▶ **Entity:** collection of characters that can be referenced as a unit
- ▶ **Element:** component of the hierarchical structure defined by a document type definition (DTD); identified by a start- and end-tag

■ Example of an entity within an element

<para>This paragraph (part of the structure for a section) references the entity **&yonder;**, and it doesn't even know where the entity is!**</para>**

■ How it "bolts together":

In document instance:

```
<!ENTITY yonder SYSTEM "<IDEA>doc2" >
:
...the entity &yonder;, and....
```

In Entity Registry:

```
<!ENTITY doc2 SYSTEM "{FileNet file id2}" >
```

Figure 3. Integration of Data, Entity Manager, and Tools with Moving & Storage

