SGML:
TECHNICAL INFRASTRUCTURE
OVERVIEW

Chris Savage
XIST Inc.

July, 1998
The IFLA Core Programme on Universal Dataflow and Telecommunications (UDT) seeks to facilitate the international and national exchange of electronic data by providing the library community with pragmatic approaches to resource sharing. The programme monitors and promotes the use of relevant standards, promotes the use of relevant technologies and monitors relevant policy issues in an effort to overcome barriers to the electronic transfer of data in library fields.

CONTACT INFORMATION

Mailing Address:

IFLA International Office for UDT
c/o National Library of Canada
395 Wellington Street
Ottawa, CANADA
K1A 0N4

UDT Staff Contacts:

Leigh Swain, Director
Email: leigh.swain@udt.ifla.org
Phone: (819) 994-6833

or

Louise Lantaigne, Administration Officer
Email: UDT@udt.ifla.org
Phone: (819) 994-6963
Fax: (819) 994-6835
Email: UDT@udt.ifla.org
URL: http://www.ifla.org/udt/

Occasional papers are available electronically at: http://www.ifla.org/udt/op/
An SGML installation is a document-processing system consisting of the technological infrastructure and tools required to create, maintain, and deliver information stored in SGML format. Every SGML application is unique, with its own content, purpose, structure, operating environment, and users. SGML installations may have common elements, but the idiosyncratic properties of the information to be managed ensures that each SGML installation will differ in terms of complexity and scale. Thus, there can be no "typical" SGML installation.

The absence of a standard SGML installation means there is no such thing as a standard SGML installation starter kit or prerequisite set of SGML development tools. SGML installations will use different tools according to: 1) the nature of the information; and 2) the complexity of requirements for manipulating the information. For each of these scenarios there are tools with varying power, functionality and costs. To determine which tools are needed to develop and sustain an SGML installation first requires analyzing the data and defining the scope of the project in terms of information use, origin of the data, development scale, and output needs. Once these issues are resolved it is possible to develop a production cycle and select appropriate tools to facilitate each function.

SGML installations can be as simple as a few SGML documents created with a public domain DTD and stored in a flat file directory on a PC, or as complex as millions of SGML documents created with a custom developed DTD and stored in a distributed document management system. Regardless of its scale or complexity, however, every SGML installation infrastructure is determined by five principal design factors:

1. Purpose of the system and intended use of the information.
2. Conformance required with existing information systems and organizational operations.
3. Costs to purchase hardware, software, train staff, develop and maintain the system.
4. Compatibility of the hardware, SGML tools, and computer operating systems.
5. Future uses and projected growth of the SGML installation.

Every organization will weigh the importance of the five principal design factors differently and devise their own custom-fitted SGML installation. Despite their differences, however, all SGML installations contain three fundamental components:

1. Human resources to develop and maintain the installation
2. Software to manipulate the data
3. Computer hardware to process and store the data.

The interaction of these components is the production cycle or operations workflow. It is how the human resources use the software and hardware to build, operate and maintain an SGML installation. Although each SGML installation will involve some operations more than others, there are three general classes of operations:

**Input** – analyzing the data; developing, customizing or adopting a DTD; inputting (composing or converting, editing, and validating) the data into the...
Processing – storing, tracking the SGML data; managing the growth of the data as it changes in time, managing workflow

Output – developing layout stylesheets to present the data in paper or electronic formats.

Depending how the five principal design factors are calculated, the scale and configuration of these operations will differ. For example, one installation may have large input and management operations to convert multi-format legacy databases into a document management system. A publisher may require a large output operation to repackage the SGML data into proprietary CD-ROM, HTML, PDF, ASCII text or other electronic formats with elaborate page layout and indexing features. In contrast, another installation may have a small output operation because its sole purpose is for long-term, information archiving. In each scenario, the SGML installation is a constructed compromise of the five principal design factors, correspondingly with a unique production cycle.

This paper provides a high-level overview of the technical infrastructure (hardware and software components) required for an SGML installation. The components of an SGML technical infrastructure fall roughly into six categories:

1. data design and validation tools
2. authoring tools
3. legacy data-to-SGML translation tools
4. document storage and management tools to support output to multiple formats.

1. DATA DESIGN AND VALIDATION TOOLS

The first stage of the production cycle is data analysis performed to determine the information requirements for the DTD, and/or the processing requirements to convert the data into SGML. Data analysis tools are used to scan original data to glean recognizable patterns in page formatting, style codes, special characters and recursive markup. These tools reduce much of the drudgery of the work, but they do not replace the need for human analysts.

Off-the-shelf data analysis tools do not exist. Instead, there are programming and scripting languages to develop customized data analysis programs. Developing these programs first requires a human analyst to perform a preliminary analysis of the data to identify information content and formatting patterns. Once patterns are known, programs can be written to search through the data and generate reports that outline the predefined data objects, formatting or style codes. It is important to note that the tools can be used to expedite the identification of formatting data but the intelligence required to determine meaningful structure is difficult, if not impossible, to automate.

There are many programming and scripting languages available to develop data analysis programs. The differences among them are extensive, and will not be outlined here. The most important criteria for selecting a program development language are:

- What training if any do the programmers require? In general, it is best to use a language that the programmers currently understand. If the programmers are experienced C++ programmers then why change to another programming language? In most cases it is more effective to use an underpowered programming language that is well understood rather than a comparatively more powerful language which requires additional training to use.

- What operating system does it need to run? The tools should integrate with current operations. If the data are formatted for Unix machines then it makes some sense that the programming language should be Unix-based because testing and debugging can be done directly in the data’s environment.

- Are the programs required to port to other operating systems? If multi-platform functionality is required, then scripting languages like Perl and Java are good choices. Compiled programs must be recompiled separately for each platform, thereby increasing the development time.

- Will the programs require continual revision? Scripting languages execute slower than compiled programs but they are faster to debug and in many cases simpler to port to other platforms.
Common programming and scripting languages to develop data analysis programs:

- Omnimon
- Spitbol
- Perl
- Sed
- Awk
- C++
- Java

1.1. DTD DEVELOPMENT TOOLS

The stage after data analysis is the development of a DTD. This may involve applying, with few or no changes, a freely available DTD found in the public domain, or developing a custom DTD. In either case, there are specific tools required to design, generate, create, view, and analyze the DTD:

- DTD design tools
- DTD generators
- DTD editors
- DTD viewing tools
- DTD documenters.

1.1.1. DTD Design Tools

A DTD can be designed with any ASCII text editor, however, to develop complex DTDs with a highly nested structure, it is better to use an editing tool that graphically depicts the DTD in recursive, colour-based, or geometrical representations. Graphical editors assist the author in tracking rules, avoiding syntax violations, and improving conceptualization. Drag and drop functionality combined with a graphical user interface improves the efficiency, speed and accuracy of designing a DTD.

1.1.2. DTD Generators

Developing a DTD is usually done manually with a tool like Near & Far Designer, but it is also common to generate DTD declarations as new information is added to the SGML application or the DTD evolves. DTD generator tools can read through documents, detect information structures or rules syntax, and generate whole or subsets of DTD declarations. They can even view a well-written XML document and generate a DTD.

1.1.3. DTD Editors

These tools are used to edit or create SGML DTD declarations. They simplify the development of SGML syntax rules by tracking DTD declarations as they are written. They are inherently customizable and useful in controlling syntax variations by presenting the DTD in a visual, user-friendly way. Some of the distinguishing features of these tools are graphical user interfaces, ability to compile the DTD into the binary format required by common SGML editors, and ability to edit declarations directly and modify the syntax throughout the DTD. The better packages include integrated syntax validators and parsers.

Experienced SGML developers do not find DTD creation especially difficult. For this reason, they do not need nor ordinarily use standalone DTD editing tools. On the other hand, novices or people who infrequently develop DTDs may find DTD editing tools helpful for visualizing and analyzing DTD declarations as they are prepared. It is important to note that the functionality of DTD editing tools can be found in other classes of tools such as DTD design tools.

1.1.4. DTD Viewing Tools

Also known as DTD browsers, DTD viewing tools simply display the DTD in a textual or graphical presentation. These tools are useful for comprehending the hierarchical structure of the DTD and the logistic effects of inclusions, recursion, and disconnected elements. Some of the features that differentiate DTD viewing tools are the number of visible elements and hierarchy levels; attribute display; attribute values display; display of inclusions and exclusions; ability to contract, expand, or zoom views; as well as generate reports. Although there are stand-alone DTD viewing tools, it is common that DTD viewing functionality is bundled with other tools capable of DTD editing or design.

1.1.5. Documenters

A DTD has two parts:

1. the **formal declarations** of entities, element types, attribute definition list, notations;
2. the informal documentation that specifies the semantics of elements, attributes and syntax conventions.

DTD documenters read the formal declarations of a DTD and generate the informal documentation. They generate reports of the skeletal structure of declaration sets, listing the permissible attributes and content for element types, depths of recursion as well as the broader definitions of syntax. These tools do not replace the need for human analysis; they assist the analyst to prepare the necessary documentation that reflects and clarifies the DTD declarations.

2. AUTHORING TOOLS

2.1. SGML Authoring tools

After the DTD is developed the data must be encoded in SGML tags according to the syntax rules defined in a particular DTD. Which type of tool is used is determined by one significant factor: does the data exist in a structured format? If it is stored in a structured format then there are conversion tools that can glean the structure from the data and migrate it into SGML format to comply with the DTD. If it is not in a structured format than it can either be input directly or marked up into SGML format using any one of several different types of SGML authoring tools.

The functionality and features of SGML authoring tools are diverse. An SGML authoring tool can be as basic as a text editor or word processor with keyboard macros defined to insert SGML elements in place of typed character strings. These kinds of tools save data entry time and simplify the creation of an SGML instance but offer little else. The current direction in SGML authoring tool design is moving towards hybrids of word processors, text editors, WYSIWYG page formatters, SGML validators, parsers, and sophisticated multimedia authoring suites. The more advanced SGML authoring tools not only mark up data with SGML tags, but can validate the syntax to ensure the output SGML instance is error-free and complies with the DTD.

The following subtypes of SGML authoring tools are examined separately:

- SGML editors
- Editors with SGML add-on

- Multimedia authoring systems
- Editing utilities
- SGML formatters
- Non-SGML formatters
- Editor / formatters with SGML editing add-on
- Editor / formatters with SGML conversion add-on.

2.1.1. SGML editors

These standalone tools are advanced text editors with macros to insert SGML markup as well as improved GUI interfaces to simplify tagging and show permissible elements and attributes. Designed specifically for editing SGML, the true strength of this type of editor is that it parses the data as it is encoded into SGML and validates the syntax against the DTD. This structure-aware feature ensures that SGML instances are syntactically correct and human error is minimized. The benefits of these tools are that they:

- simplify the editing process by reducing keystrokes to insert markup with pull-down menus, buttons, and floating palettes
- produce error-free markup through validation
- offer a simple-to-operate user interface.

Some of the common features in SGML editors are:

- context-sensitive tagging (e.g. only permit the selection of valid elements and attributes)
- built-in parsers
- inter-operable with any DTD
- on-the-fly or user-invoked parsing and validation
- multiple, simultaneous documents, records, entities and fragments editing
- graphics handling (permit referencing graphics entities and in some cases inline display)
- tables and math formulas support
- draft mode printing capability (although limited and not a replacement for FOSI editors)
- customizable application programming interface (API) or dynamic link libraries (dll) integration to adapt the editor to the needs of the author
- word processing features such as cut/copy/paste, search and replace
- content validation such as spell checking, grammar checking, thesauri
- minor page and style sheet formatting
- support for extended character sets
- support for HyTime and DSSSL standards
• output to native SGML (in contrast to proprietary binary formats).

Their disadvantages include limited capabilities for formatting, composition and printing. Also, some SGML editors cannot output to native SGML, making them difficult to integrate into an SGML implementation unless the other tools are capable of reading the editor’s proprietary file format.

There are two major classes of SGML editors, native and structured. The difference between them is how they manipulate and store the data. Native SGML editors import, manipulate and store the data in SGML format (ASCII text). In contrast, structured SGML editors use their own internal formats. Structured SGML editors filter and convert native SGML and DTDs as they are imported and exported. The disadvantage of structured SGML editors is they typically do not provide as extensive validation or feedback as instances are authored, and they are not as easily integrated into an SGML installation with other tools because of their proprietary file formats.

2.1.2. Editors with SGML add-on

In addition to SGML editors that are designed specifically for handling SGML, other text editing and composition tools can also be used to author SGML with the aid of add-on macro routines. Also called write-to-convert tools, these add-on routines make the base program “SGML aware” and provide some measure of SGML structural enforcement. They operate by mapping formatting instructions from the editor to SGML markup instructions. For example a keystroke combination that would otherwise format a block of text as bold could be mapped with the instruction to mark the text as a subtitle.

There are two categories of tasks these tools perform: editing and conversion. Editing with an add-on can provide some control over structural validity, although not as strictly as an SGML editor can. Conversion involves importing a document with styles or a structure and outputting to SGML. Whereas editing can offer a degree of control over validity, conversion offers even less. To compensate for this weakness, some of these tools can integrate with an external parser and produce error reports so that the author can reopen an instance and make the corrections manually.

The advantages of these tools are that they allow the authors to continue using the tools with which they are skilled, and that they are usually much less expensive. The disadvantages are: the validation is not as strong as SGML editors; they usually do not integrate as well with the other SGML tools; and they tend not to be the most up-to-date for new features and enhancements. Since they are add-ons to software packages designed for other purposes, it is also likely that they will not be revised as frequently as the base program. This means it is possible that the add-ons will only function with older versions of the base program than the SGML author has.

2.1.3. Multimedia authoring systems

Multimedia authoring systems are still very immature as a technology and should continue to evolve in the near future. This type of tool provides the ability to create SGML instances that combine SGML text and binary objects such as graphics, audio, and video files. The challenge with multimedia authoring is maintaining the structure of the SGML documents. Some tools of this type can import SGML documents but most will flatten all SGML abstract information as it is saved into a proprietary non-SGML notation. A good multimedia authoring tool will maintain the links and specifications of semantics between objects, permitting them to be reused in other SGML instances.

2.1.4. Editing utilities

Since native SGML is ASCII text, general-purpose text editors can be used to manipulate SGML data. Markup is typed in manually without convenient pull down menus, keyboard shortcuts and buttons to expedite the process. These editors do not have built-in parsers and validators so there is no error checking functionality. For this reason, authors must know the DTD well, as the potential to create invalid markup is immense. External parsers can be used to verify the markup but this involves an extra step that SGML editors have built-in.

The advantages of using a general editing utility are:

• convenience – every platform has at least one text editor utility; they are fast to load, simple to operate; useful for making quick changes.
• control – the author can enter temporary invalid code for short references and tag omissions that most SGML editors will not permit.

Disadvantages include:

• no built-in parsing and validation
• no content verification such as grammar and spell checking
• not suited for developing multiple, simultaneous SGML instances.

2.2. Formatters

Formatters are used to develop a paper presentation of an SGML instance by associating formatting instructions to the SGML element structure. They create renditions of the SGML abstraction with or without changing the underlying SGML code. SGML formatters are the functional equivalents of word processor style sheets or formatting templates. They translate SGML markup into font types and sizes, as well as placement on the printed page, inclusion of headers, footers, page numbers, navigation, cross-references, footnotes, endnotes, table of contents and indices, etc.

There are two different approaches to developing a formatting scheme:

• **Grove-based** – the formatter reads the element structure and formats the document relative to the hierarchy structure.

• **Markup-based** – the formatter maps formatting codes to specific markup elements.

Grove-based formatters are the simplest to use and most powerful for managing large collections of SGML data because they create formatting style sheets that can be easily applied to different SGML instances. Mark-up based formatters are best used to exercise tight control over the appearance of specific SGML instances. They do not transfer well to broad sets of SGML instances because they are too closely modeled on the structure and markup of discrete SGML instances.

Features to look for:

• Formatting style sheets can be saved separately from the document so they can be reused and applied to other SGML instances.

• Multiple style sheets can be developed and stored so an SGML instance may have different presentations.

• Content can be generated, suppressed and reordered (e.g. pagination altered, chapters renumbered, subtitles and captions suppressed).

• Style sheets can be saved in native SGML notation so that other management tools can manipulate the style sheet instructions.

• Style sheets can be saved in standardized specifications such as DSSSL and FOSI, so they can be understood and manipulated by editors, browsers and other formatters that support the standard.

2.2.1. SGML formatters

The distinguishing feature of these formatters is they produce style sheet renditions directly from the SGML abstraction. They have built-in parsers that differentiate the markup from content and facilitate syntax validation. The fact that they manipulate the data in native SGML means that changes to the content and syntax can be made directly to the source documents. This reduces the problems associated with version control.

2.2.2. Non-SGML formatters

These formatters produce renditions in proprietary, non-SGML formats. They operate by down-conversion (importing the SGML source instance, and converting it into their own format). The limitation of this type of formatter is it cannot directly edit the SGML code. To edit the SGML code after formatting, the rendition must be converted back into SGML. In most cases this extra step loses much of the original SGML code such as SGML comments and sections of markup the formatter had ignored. The alternative is to reopen the SGML source document, make the necessary edits, and down-convert the document back into the non-SGML formatter. Although the source document’s integrity is maintained in this scenario, the formatter will have to redo all the formatting that was previously done. For these reasons non-SGML formatters are best suited for producing one-off style sheets that are discarded after use.

2.2.3. Editor/formatters with SGML editing add-on

These tools are third-party macros that plug into common office applications such as Microsoft Word.
or Corel WordPerfect. They change the behaviour of the base tool so that it acts like a native SGML editor. They have basic SGML-awareness with limited built-in parsers and validators, but they are not as well integrated as SGML editors. Their advantage is they extend the functionality of existing tools and are typically less expensive than full SGML editors. They are useful for small SGML development projects; well suited for designing output renditions or stylesheets of simple documents; and integrate well with DBMS systems. Their disadvantage is slower performance, weak SGML-awareness, and add-on upgrades may not keep pace with upgrades of the base editor.

2.2.4. Editor/formatters with SGML conversion add-on

These tools are similar to editor / formatters with SGML editing add-on, except they only map SGML notation to the base tool's proprietary formatting codes. They are not suited for building interactive SGML collections that require editing or continual revision. The key is they are designed to import SGML notation, but not export selected parts. They can export a whole document, if the filters are simple. For this reason they are poor choices as an exclusive editing tool.

2.3. Parsers and Validators

Parsers are programs that distinguish the SGML markup from the content of a document. Parsers are usually bundled with other tools such as SGML editors, converters, and data management tools. As such it is not common to purchase a parser as a stand-alone tool. Validators, also called validating parsers, are parsers with the added capability of comparing the SGML markup against a DTD and reporting errors.

3. LEGACY DATA CONVERSION TOOLS

Legacy conversion involves translating existing data from one information system or file format into SGML format. The first challenge is to determine the structure of the legacy data; the second is to map it to an SGML DTD. There are two types of source data to convert: SGML and non-SGML, such as word processing or database formats. S-converters adapt SGML data to other SGML DTDs (called "down" conversion), whereas N-converters convert non-SGML data into SGML structures (called "up" conversion).

3.1. SGML Legacy Data

3.1.1. Determining structure in SGML documents

The main types of down conversion processes are:

- Parse the source SGML of a whole document and convert to a new DTD
- Extract portions of an SGML document, parse and convert to a new DTD
- Merge SGML documents together, parse and convert to a new DTD.

In all these three types of conversion the SGML must be parsed to determine the information structure and elements.

Unless it is poorly coded, detecting the structure of an SGML document is relatively straightforward. The complication is mapping the source SGML accurately to the new DTD. Although it is simpler to convert a detailed SGML source to a broad SGML DTD than vice versa, too much detail can create difficulty. Cross-references can require analyzing multiple documents to convert properly. Not only is this taxing for the hardware, in some cases requiring extensive memory to load all related documents, but it can also prove difficult to determine the logical structure of the collection.

3.1.2. Down Conversion Tools

Source SGML data can be converted from a variety of formats and storage systems. It can be stored as whole SGML documents, style-based documents in flat directories, or SGML fragments in databases. A number of conversion tools exist that can interface with ODBC compliant databases to convert either SGML or non-SGML data. Since this data is fielded, it is simple to parse the fields into a document structure and convert to an SGML DTD. The significant challenge is determining the logical structure of data within the fields. Conversion tools not only can translate the data fields to a DTD, but they are programmable and intelligent enough to parse the structure within fields. In contrast, the host database application can also export the data inside wrapped SGML code, though it is far more difficult to parse the data within fields. In addition, mapping the structure of the DTD is more tedious and error prone with the host database application than it is using a tool designed specifically for converting database data to SGML.
A cross between SGML converters and parsers are markup translators. These tools parse SGML, expand or edit the markup to comply with other SGML DTDs. They are typically not as powerful as SGML converters but helpful for preprocessing SGML documents so they can be integrated into other SGML production processes. Common functions are to add omitted tags (e.g. start or end tags), replace references, extend attribute specifications, change tag cases, and add or remove delimiters.

3.2. Non-SGML Legacy Data

3.2.1. Determining structure in non-SGML documents

The most common way to determine structure in non-SGML documents is to analyze the formatting codes. Typographic formatting is used to improve the reading of a document as well as indicate a logical structure, and convey information about the data. For example, document titles are often the largest font type in a document. The type size and font indicates a logical relation to the other parts of the document. Other formatting styles like italics convey additional meaning about the text, such as the text is an undefined term, publication title, or emphasized, etc.

There are several challenges to determining structure in non-SGML documents:

- Formatting codes are commonly inconsistent. Not only are they frequently inconsistent within a document, but also from one document to the next. This makes batch conversion of a collection difficult.
- Formatting codes commonly indicate multiple meanings in the same document. For example, in one place a bold font may indicate the text is a subheading (significance related to the document), in another place it may indicate it is a proper noun (significance related to the term).
- Data can be complex and nested. Formula data that is presented in tables, for example, is difficult to batch convert with accuracy.
- Human error in the original documents is difficult to filter through conversion tools. New technologies such as word processors encourage stylesheets for formatting documents and promote improved logical structuring of data, but older applications did not. Legacy data generated in antiquated technologies may have insufficient formatting codes wrapped around the data to filter into logical significance.
- Conversion involving scanning from paper to electronic formats is especially difficult.

Other than detecting structure by parsing formatting codes, it is also possible to search for natural language strings. For example number or bullet lists and annotations such as "illustration" or "see also".

3.2.2. Up Conversion Tools

Conversion tools are programmable to search for patterns matching text-based expressions and conventions. Database applications have limited text string search and replace capabilities. This means it is possible to search formatting codes and natural language within database fields, but they are not as powerful or intelligent as SGML converters.

4. DOCUMENT STORAGE AND MANAGEMENT

The main processing operations of an SGML installation are storage, retrieval and file maintenance. There are several classes of technologies to perform these tasks and manage various sizes and complexities of SGML collections.

The simplest are the file management systems built into every operating system. These tools store SGML instances as separate, unrelated files. Although inexpensive, they offer only basic security and are limited for managing dynamic, compound documents that integrate many SGML instances of different granularity. Better alternatives for handling compound documents are database applications (DB) because they can store SGML instances as separate but related components. Their weakness is they do not have any built-in SGML awareness or advanced record management features. Depending if it is a relational, object-oriented or object-relational DB, they store SGML data simply as fields, records, objects or binary large objects (BLOBS). The DB builds indices that track the location of the data, but is blind to the SGML notation, reading the SGML code strictly as text strings. This recurrent translation process, from SGML document structure to proprietary database index structure and back to SGML output, can also lead to a loss of SGML information, particularly with highly nested SGML code. Such a loss is attributable to the differences between the data...
models of SGML (multifaceted) and DB technologies (inherently hierarchical and tabular).

**Database management systems (DBMS)** are an alternative to DB technologies. These technologies incorporate a database backend to store SGML instances as separate parts or whole documents, but also have integrated features such as secure access, revision history control and tracking, and powerful searching capabilities. Better yet, but more expensive, are **document management systems (DMS)**. DMS are like DBMS but with an additional layer of middleware that can map the SGML DTD and document components directly to the database, some without saving separate proprietary format indices. A DMS may also offer: workflow management; link management; concurrent access to document components by multiple users; additional SGML validation; table conversion; reports generation for both data and SGML notation; advanced text sorting; SGML-aware searching on elements, attributes, entities, links, sections; and Boolean, proximity, wildcard, keyword in context, and full-text searching. Better yet, but more expensive, are **document management systems (DMS)**. DMS are like DBMS but with an additional layer of middleware that can map the SGML DTD and document components directly to the database, some without saving separate proprietary format indices. A DMS may also offer: workflow management; link management; concurrent access to document components by multiple users; additional SGML validation; table conversion; reports generation for both data and SGML notation; advanced text sorting; SGML-aware searching on elements, attributes, entities, links, sections; and Boolean, proximity, wildcard, keyword in context, and full-text searching. Better yet, but more expensive, are **document management systems (DMS)**. DMS are like DBMS but with an additional layer of middleware that can map the SGML DTD and document components directly to the database, some without saving separate proprietary format indices. A DMS may also offer: workflow management; link management; concurrent access to document components by multiple users; additional SGML validation; table conversion; reports generation for both data and SGML notation; advanced text sorting; SGML-aware searching on elements, attributes, entities, links, sections; and Boolean, proximity, wildcard, keyword in context, and full-text searching.

All four types of information management technologies can support SGML collections, but each has strengths particular to different needs and purposes. File management systems and DB applications are best suited for small collections that do not require extensive file maintenance, revision of SGML instances, output capabilities, or advanced searching. They have high fault tolerance, low (if any) SGML awareness, and can be expensive to modify as collections grow, needs change, and additional functionality is required. Integration with other development tools can also be a problem, as they are not designed specifically for SGML publishing systems. For this reason any SGML installation which uses a DB to manage data will require customized scripting or API programming to integrate with other tools. Depending on the required functionality of the system and mix of tools, the costs to customize the system to integrate the parts may exceed that for an integrated, off-the-shelf DBMS or DMS. These types of tools offer many packaged scripts and programs to manage SGML data that can be used with little or no modification.

The most important consideration for selecting processing technologies is how they integrate with the other parts of the SGML installation. Small collections with limited growth potential do not need elaborate DBMS or DMS technologies. It is possible to build functional SGML installations with a basic input tools such as a word processor that map a DTD to a “style” structure and stores the data in the operating system’s built-in file management system. Replacing the file management system with a DB can improve searching and management but it may require some middleware to integrate the authoring tool. The key point is that the required functionality determines the tools, and the tools in turn raise new questions regarding integration compatibility.

**5. OUTPUT TO MULTIPLE FORMAT TOOLS**

Many different types of tools can output SGML data to a variety of electronic and print formats. It is common for editors, formatters, page composition, and document storage and management tools to be able to produce SGML renditions. Choosing the appropriate tool depends on numerous related factors such as:

- Complexity of the desired output
- Output media and document delivery
- Number of renditions or styles required for the same SGML data (e.g. separate versions for CD ROM, paper, HTML, Braille, voice synthesis and audio, etc.)
- Complexity and integration of the publishing system (e.g. document creation and management systems)
- State of the SGML document (e.g. is it stored as a whole document or assembled on the fly?)
- Style specification language (e.g. FOSI, DSSSL, XML)
- Compatibility with other tools.

Since each SGML installation has different needs and configurations of technologies, a standard output tool that suits all does not exist. The potential to resolve the above mentioned factors differently means, for example, that while the best output tool for one installation is a structured editor, for another it may be a multimedia authoring tool. Each output tool is a compromise of needs and functionality. WYSIWYG editors are popular because of their
simplicity of use. They are well suited for designing style sheets for print renditions because they operate much like word processors. Balanced against this feature, though they may be a poor choice for large systems built around a DMS with complex SGML DTD mappings.

NOTES

Much of the research and analysis is derived from:
