ELECTRONIC PUBLISHING, VOL . 5(3), 105–130 (SEPTEMBER 1992)

On the interchangeability of SGML and ODA

CHARLES K. NICHOLAS[1] AND LAWRENCE A. WELSCH

Office Systems Engineering Group
Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA

SUMMARY

SGML and ODA are international standards for the markup and interchange of electronic documents. These standards are incompatible, in the sense that in general a document encoded using SGML cannot be used directly in an ODA-based system, and vice versa. We first describe these two standards, and suggest criteria under which a bridge between the two standards could be evaluated. We then evaluate the Office Document Language (ODL), an SGML application specifically designed for ODA documents, with respect to these criteria. We describe conditions under which reliable automatic translation between SGML and ODA can be achieved, and describe a translation program that converts SGML documents to ODA and back.

KEY WORDS SGML ODA ODL Document interchange

INTRODUCTION

The Standard Generalized Markup Language (SGML) [1] and the Office Document Architecture (ODA) and Interchange Format [2] are the two most prominent standards for the markup and interchange of electronic documents. Considerable debate has ensued over the relative strengths and weaknesses of these two standards. Although they both address the problem of electronic document interchange, they differ in their approach to this problem, and in the functionality they provide to solve it.

The existence of two standards places those with interest in the document interchange problem in the unfortunate position of having to choose a standard, and thereby risk choosing the wrong standard, or accepting both (to some extent) and putting up with a duplication of user expertise and investment in interchange software. Communities of users have grown up around each standard, as well as companies committed to serving the software needs of those users. Any attempt by a government agency or standards organization to establish a single interchange standard by fiat will probably fail. Since it is likely that both standards are going to remain in use for some time to come, it is appropriate to consider ways in which the duplication of effort costs can be minimized, and to find ways in which users of each standard can take advantage of the features offered by the other.

One way to reduce these costs and permit users to take advantage of both standards is to develop a mechanism for moving documents from one standard to the other.

[1] Also affiliated with the Computer Science Department, University of Maryland Baltimore County, Baltimore, MD 21228-5398, USA. Electronic mail: nicholas@cs.umbc.edu. Contribution of the National Institute of Standards and Technology. Not subject to copyright.

CCC 0894–3982/92/030105–26
Received 7 August 1991
Revised 12 August 1992

Furthermore, this mechanism should be as automatic as possible. Such a mechanism can then be regarded as a bridge between the two standards. If such a bridge were available, then

- Users of one standard would not need to learn the other in order to import or export documents encoded using the other standard. This would minimize duplication of effort in terms of user expertise.
- Developers of tools would be able to address the needs of a single market without significant additional investment.
- The purpose of both standards, document interchange, would be better served.

Reliable automatic translation between compatible SGML and ODA documents can be achieved under certain conditions. We begin with overviews of SGML and ODA, using an early version of this article as a running example. Then we propose some criteria that a bridge between SGML and ODA should satisfy, and assess the Office Document Language (ODL) with respect to these criteria. We then describe how reliable automatic translation between SGML and ODA can be achieved when these criteria are satisfied. Finally, we describe the software we built to demonstrate the ideas presented in this paper.

AN OVERVIEW OF SGML

The Standard Generalized Markup Language, or SGML, is a language and notation for describing classes of documents. In an SGML-encoded document, such as the document shown in Figure 1, the various *elements* of the document are delimited with distinguished character strings commonly called *tags*. A document may, for example, have tags that delimit elements like paragraphs, subsections, appendices, and figures. A *start-tag* of the form <X> denotes the beginning of an element, and an *end-tag* of the form </X> denotes the end of that element.

An SGML-encoded document has three parts: an *SGML declaration*, a *document type declaration*, and a *document instance*. The SGML declaration characterizes the document type declaration, and the subsequent document instance(s), in terms of character sets and optional features of SGML. The SGML declaration may be omitted if the default character set is used and no optional features are required.

In SGML, a class of documents is characterized by a grammar that indicates what markup is allowed, what markup is required, and how markup is distinguished from text. SGML defines this grammar with a *document type definition*, such as the one shown in Figure 2, that describes a class of documents in much the same way that formal grammars (written in BNF, for example) describe programming languages. In particular, document type definitions, or DTDs, describe how elements may be used in a document. SGML itself does not specify any particular set of elements; the set of elements that can be used in a document is specified by its document type declaration. The document type declaration may invoke a public document type definition, introduce an original document type definition (such as the one shown in Figure 2), or invoke a public DTD and add extra elements to it.

```
<ARTICLE>
<TITLE>
On the Interchangeability of SGML and ODA/ODIF
<AUTHOR>
Charles K. Nicholas
<AUTHOR>
Lawrence A. Welsch
<AUAFF>
Office Systems Engineering Group
Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899
<ABSTRACT>
<PARA>
SGML and ODA/ODIF are international standards for the markup and
interchange of electronic documents.  These standards are
incompatible, in the sense that a document encoded using SGML cannot
be used directly in an ODA-based system, and vice versa.  We
first describe these two standards, and suggest criteria under which a
bridge between the two standards could be evaluated.  We then evaluate
the Office Document Language (ODL), an SGML application specifically
designed for ODA/ODIF documents, with respect to these criteria.
</PARA>
<SEC><SECHEAD>
Introduction
<PARA>
The Standard Generalized Markup Language (SGML)
<ENR>
Information Processing SGML
</ENR>
and the Office Document Architecture/Office Document Interchange
Format (ODA/ODIF)
<ENR>
Information Processing ODA
</ENR>
are the two most prominent standards for the markup and interchange of
electronic documents.  Considerable debate has ensued over the
relative strengths and weaknesses of these two standards.  Although
 .
 .
 .
```

Figure 1. Part of an SGML-encoded document. The first several lines of an early draft of this article, marked-up in accordance with the SGML document type definition (DTD) shown in Figure 2

The general form of an element declaration is

```
<!ELEMENT name    omittag-info      content-model>
```

where *omittag-info* indicates whether start-tags and end-tags may be omitted, and *content-model* describes the data or additional markup that may appear within the element.

The DTD in Figure 2 includes definitions for twenty-eight elements. The article element, for example, is delimited by the <ARTICLE> start-tag and the </ARTICLE> end-tag. The two dashes in the omittag-info field indicate that the start-tag and the end-tag must be present. The dash in the definition of para indicates that the start-tag must be present, and the “O” indicates that the end-tag may be omitted. The content-model field

```
<!DOCTYPE ARTICLE[
<! Articles contain frontmatter, a body, and maybe some appendices. >
<!ELEMENT ARTICLE       - -     (FRONTM, BODY, APPENDIX*) >
<! Front matter includes the title, author information, and abstract. >
<!ELEMENT FRONTM        O O     (TITLE, AUGROUP, ABSTRACT) >
<!ELEMENT TITLE         - O     (#PCDATA) >
<!ELEMENT AUGROUP       O O     (AUTHOR+, AUAFF)+ >
<!ELEMENT AUTHOR        - O     (#PCDATA) >
<!ELEMENT AUAFF         - O     (#PCDATA) >
<!ELEMENT ABSTRACT      - O     (PARA+) >
<! The body contains one or more sections, and maybe some endnotes. >
<!ELEMENT BODY          O O     (SEC+, ENDNOTES?) >
<! Each section has a heading, and text in paragraphs or subsections. >
<!ELEMENT SEC           - O     ((SECHEAD, PARA*, SUBSEC*)) >
<!ELEMENT SECHEAD       - O     (#PCDATA) >
<!ELEMENT SUBSEC        - O     (SSECHEAD, PARA+) >
<!ELEMENT SSECHEAD      - O     (#PCDATA) >
<! Paragraphs contain text, emphasized text, lists, footnotes, etc. >
<!ELEMENT PARA          - O     (#PCDATA | EMTEXT | BULIST | NUMLIST |
                                 BOLDTEXT | FN | FNR | ENR | FIGURE)+ >
<!ELEMENT EMTEXT        - -     (#PCDATA) >
<!ELEMENT BULIST        - O     (ITEM+) >
<!ELEMENT NUMLIST       - O     (ITEM+) >
<!ELEMENT ITEM          - O     (PARA+) >
<!ELEMENT BOLDTEXT      - -     (#PCDATA) >
<!ELEMENT FN            - -     (PARA+) >
<!ELEMENT FNR           - -     (#PCDATA) >
<!ELEMENT ENR           - -     (#PCDATA) >
<! Figures contain textual or non-textual information, and an optional
caption. >
<!ELEMENT FIGURE        - -     ((CODE | GRAPHIC), CAPTION?) >
<!ELEMENT CAPTION       - O     (PARA) >
<!ELEMENT CODE          - -     (#PCDATA) >
<! NDATA refers to information not encoded in the character set given
in the SGML declaration.  (This example uses the default character set.) >
<!ELEMENT GRAPHIC       - -     (#NDATA) >
<!ELEMENT ENDNOTES      - O     (ITEM+) >
<!ELEMENT APPENDIX      - O     (APPHEAD?, PARA+) >
<!ELEMENT APPHEAD       - O     (#PCDATA) >
<!ENTITY lt    "<" >
]>
```

Figure 2. An SGML document type definition for simple articles. This DTD describes the various tags, the order in which they may appear, and whether they can be omitted

contains information regarding the nesting of elements in the form of a rule, and may list exceptions to that rule, known as *inclusions* or *exclusions*. The content model for the article element, for example, indicates that the <ARTICLE> start-tag must be followed by some front matter, the body of the article, and zero or more appendices. The symbol #PCDATA indicates a place where parsed character data (i.e. ordinary text with no embedded markup) may appear.

SGML provides several other types of markup, in addition to elements delimited by start-tags and end-tags. For example, a document may contain certain character strings, known as *entity references*, that are replaced with designated values when encountered during the processing of the document. The DTD shown in Figure 2 has one entity reference, named “lt”, that should be converted to a less-than sign when encountered in a document.

An SGML *document instance*, such as the one shown in Figure 1, is processed by parsing the document and performing appropriate actions, such as the generation of corresponding typesetter commands, when the various elements are recognized. There are two mechanisms in SGML for specifying the meaning of individual elements, in terms of processing to be performed, in the DTD:

- application-specific *processing instructions* may be associated with the various elements in the DTD, and
- markup substitution, i.e. the replacement of elements or entity references by other markup or text, can be specified using the SGML *link* feature.

DTDs that describe complex, highly structured documents, such as technical manuals for large systems, may have dozens of elements for the different components of the document [3]. Several organizations, such as the Association of American Publishers [4] and the U.S. Department of Defense [5], have developed document type definitions with well-defined meanings for entity references and elements. (We have only touched on SGML's most important features. For a complete treatment of SGML, the reader is directed to Goldfarb's *SGML Handbook*.) [6]

AN OVERVIEW OF ODA

The Office Document Architecture and Interchange Format was designed to facilitate the interchange of documents in a heterogeneous network [7], [8]. The ODA encoding of a document includes information on the structure and content of the document, as well as information on its appearance when rendered on a printed page or other output media. For this reason, an ODA document is said to have a logical view and a layout view, as shown in Figures 3 and 4. These views correspond to trees, the leaves of which are associated with individual portions of the document's content.

The tree corresponding to the document's logical view is called the *specific logical structure*, and the root of this tree is the *document logical root*. The *subordinates*, or descendants, of the document logical root may be of two types: *composite logical objects* and *basic logical objects*. The subordinates of composite logical objects may be other composite logical objects or basic logical objects. A basic logical object may be the parent of zero or more *content portions*. An ODA document may have a *generic logical structure*, corresponding roughly to the DTD in an SGML document, that indicates for a given composite logical object which objects may appear as subordinates of that object. (We describe the ODA notion of generic logical structure in more detail later in this section.)

As an option, a logical object can be labeled with a sequence of ASCII characters known as the user-visible name attribute. Figure 3 shows the user-visible names of some basic and composite logical objects for an ODA document equivalent to the SGML document shown in Figure 1. The user-visible names chosen for each basic or composite logical object in the ODA document correspond to a tag in the SGML document, except for the basic logical object TEXT, as explained below.

The layout view is represented, as shown in Figure 4, in the form of a tree called the *specific layout structure*, rooted at the *document layout root*. The tree is shown with the root at the bottom to emphasize that the content leaves are the same content leaves shown

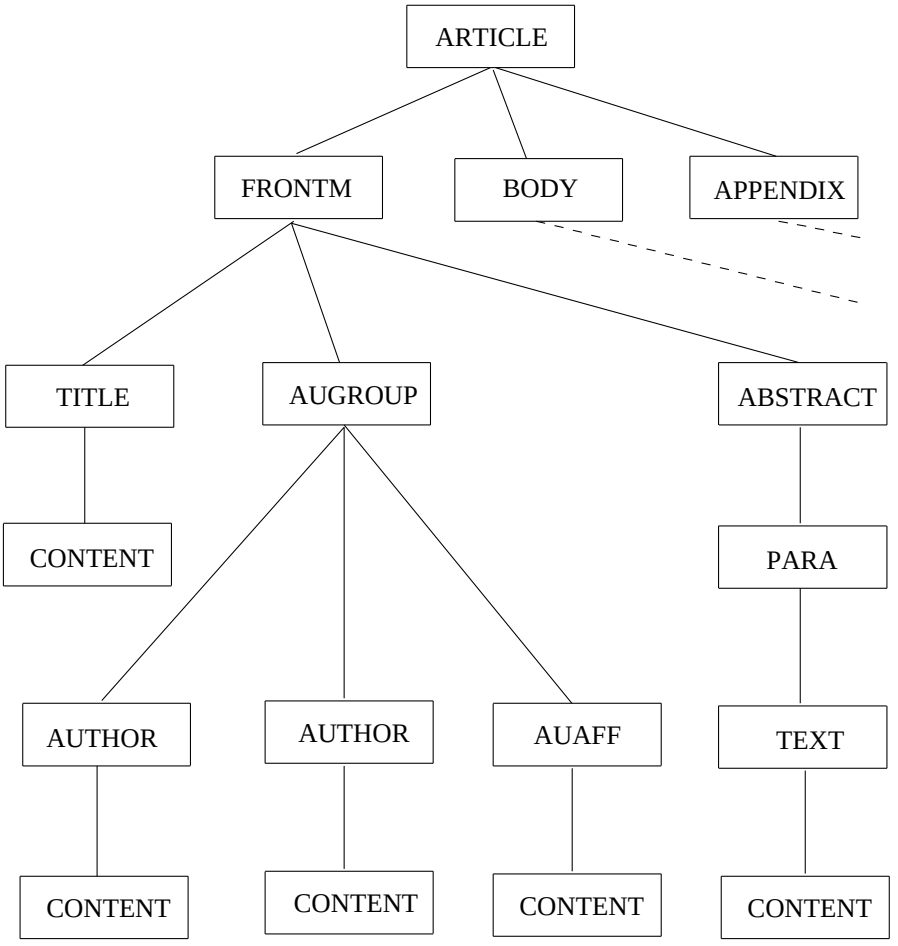

Figure 3. Part of the logical view of an ODA document. The logical structure indicates the document's organization in terms of, for example, front matter, section and appendices

at the bottom of the specific logical structure in Figure 3. The interior nodes of the specific layout structure represent specific portions of a document as it appears on an output media. Subordinates of the document layout root are layout objects. The types of ODA layout objects include *page set*, *composite page*, *basic page*, *frame* and *block*. Composite pages may contain (possibly nested) frames, and frames contain blocks. Individual content portions are associated with basic pages, or blocks within composite pages.

An ODA document belongs to one of three *document architecture classes*: the *formatted* document class, the *processable* document class, or the *formatted-processable* document class. A formatted form document is described in terms of the layout view. A processable form document is described in terms of the logical view. A formatted-processable form document, such as the document represented in Figures 3 and 4, is described in terms of both views.

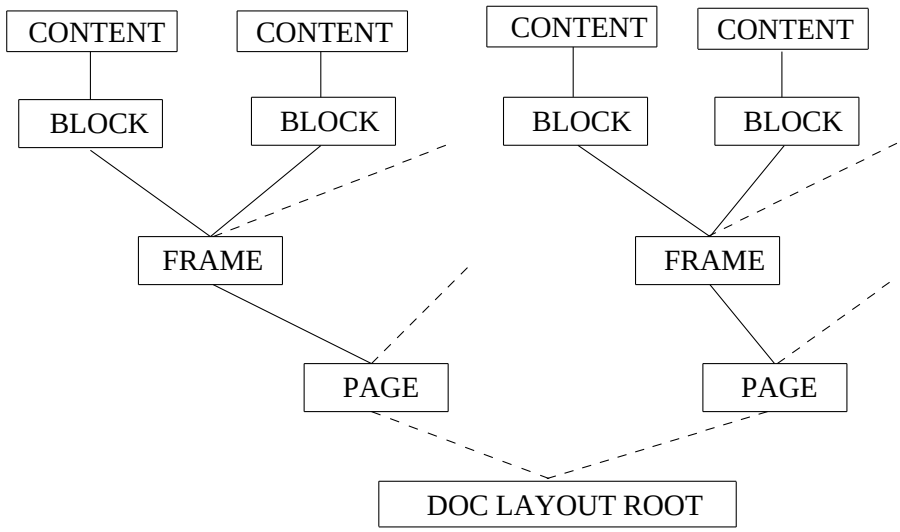

Figure 4. The layout view of an ODA document. An ODA document has a layout structure expressed in terms of blocks, frames, pages, and page sets

Content portions may contain: text; raster, facsimile or bit-mapped graphics information; or geometric graphics information in CGM format. Content portions may include special characters and other information that says, for example, how the content is to be rendered on an output device. Character, raster or bit-mapped graphics, and geometric graphics must conform to the ODA *content architectures* described in volumes 6, 7 and 8 of the ODA standard, respectively. (Additional volumes describing other content architectures are in various stages of preparation.)

Although character content in an ODA document roughly corresponds to the SGML notion of #PCDATA, there are two differences worth noting. First, ODA character content may have embedded control codes that describe how the document is to be formatted. Such formatting information would either be omitted in an SGML encoding, or transmitted with a particular tag. Second, ODA does not allow character content (or any other kind of content) to be the immediate subordinate of a composite logical object. For this reason Figure 3 shows a basic logical object called TEXT that doesn't correspond to any element in the SGML DTD in Figure 2. In the ODA version of the example document, ordinary text within a paragraph is associated with one or more TEXT basic logical objects, which are in turn subordinates of a PARA composite logical object.

In ODA, an *attribute* is a keyword and associated value that indicates some characteristic of that constituent or a relationship between that constituent and another constituent. There is a certain set of attributes associated with every constituent of an ODA document. For example, every specific logical object has an object identifier and an object type. Every composite object has a subordinates attribute that lists the objects that appear below that composite object in the logical or layout view. The subordinates attributes of composite objects in the specific logical structure show how a specific document is made up, in the same way that the nesting of tags describes the composition of SGML documents.

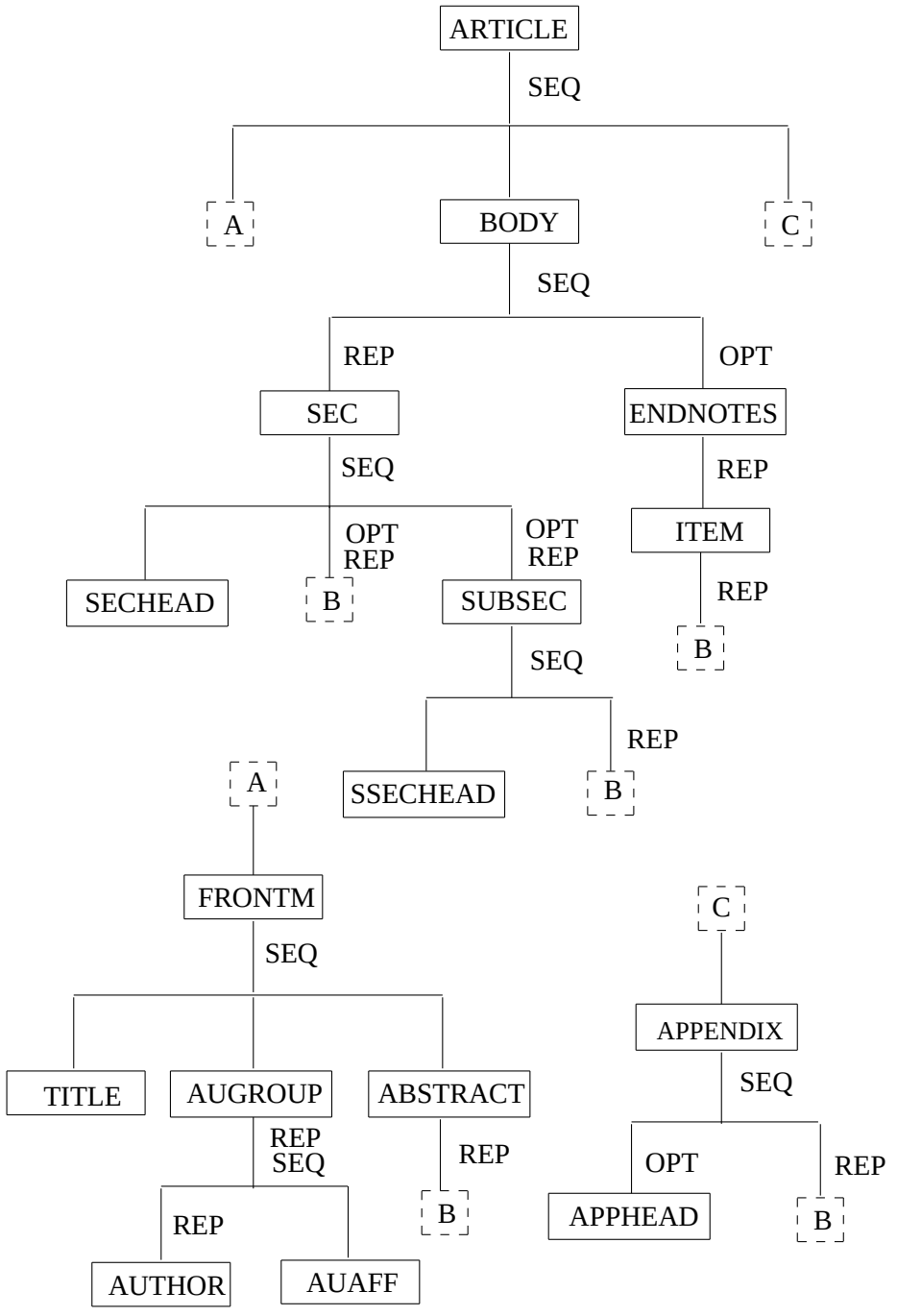

Figure 5. The ODA generic logical structure of the example document. This generic logical structure is structurally compatible with the SGML DTD in Figure 2

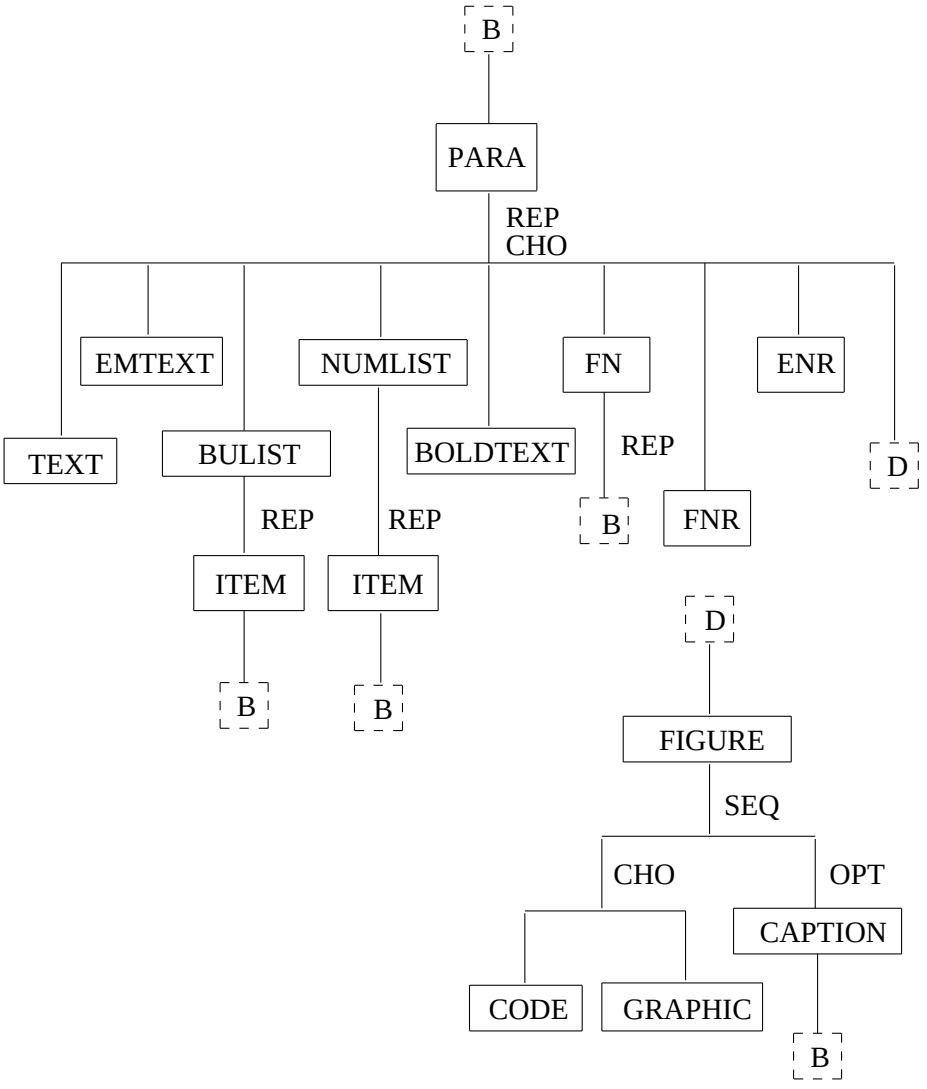

Figure 6. An ODA generic logical structure (continued). Paragraphs may contain various kinds of text, figures, and references

The various constituents of a document may have certain attribute values in common. These attributes may refer, for example, to specific formatting concerns such as indentation or point size. ODA provides two ways to specify the values of attributes for a number of objects in the same document:

- First, ODA has an elaborate mechanism for specifying default values of attributes. For example, the value of an attribute for a composite logical object may be used as the default value of that attribute for all composite and basic logical objects occurring below that object in the specific logical structure.

- Second, ODA allows a document to contain *generic logical objects* and *generic layout objects*. The attributes of generic objects need be specified only once, and may then be used repeatedly throughout a document. Once generic objects are defined and have values set for their attributes, a specific (logical or layout) object may declare itself to be a member of the class of objects defined by a given generic object, thereby inheriting that generic object's attributes.

The set of generic logical objects in an ODA document makes up that document's *generic logical structure*, or GLS. The generic logical structure of an ODA document is similar to the document type definition for an SGML document, in that the generic logical structure shows how each part of the document is composed of smaller parts. Specifically, a composite object in the generic logical structure may have a *generator for subordinates* attribute that says, using a BNF-like notation, which types of object may appear as subordinates of that composite object. The generator for subordinates attribute is therefore similar to the content model of an SGML element, which lists the elements that may be nested inside that element. Generic logical structures are also similar to SGML DTDs in that a GLS can describe a class of documents, the members of which have the same structure but different content. The generic logical structure of the example document is shown in Figures 5 and 6, in the notation defined in Part 2 of the ODA standard. The Appendix shows a portion of the ODA example document, including the generic and specific logical structures.

When a processable-form ODA document is to be rendered on a printed page or other output medium, the objects in the specific layout structure need to be created. The ODA system performing this rendering may use *layout styles* associated with the document's specific or generic logical objects to construct the layout objects. The layout of the document's content portions can be specified through *presentation styles*. An organization may build customized generic logical and layout structures for its own use, with predefined content portions of various types and house layout and presentation styles.

COMPARING SGML AND ODA

The most important difference between SGML and ODA is their different stance towards the structure and processing of documents. SGML is a language for describing the structure of documents. The processing that occurs after a document is created is not addressed directly. In particular, the appearance of the document, before or after interchange, is not explicitly considered in SGML. At the moment, SGML incorporates no formatting information, although a complementary standard called DSSSL (Document Style Semantics and Specification Language) will be used to express formatting information. DSSSL is currently a Draft International Standard, with projected completion late in 1992 [9]. Subsequent processing may be indicated with the SGML processing instruction syntax, which allows commands for some post-processor to be embedded in SGML markup.

ODA is less insistent than SGML about explicit description of document structure. ODA documents are self-describing, in two senses:

1. As mentioned above, a given specific logical object has a subordinates attribute that lists the basic and composite logical objects which are that object's immediate descendants.

2. The subordinates of generic logical objects may be described using the generator for subordinates attribute.

The generic logical structure of an ODA document corresponds to the SGML notion of document type definition. ODA has a broader model of document processing than does SGML, a model that includes creation, editing, and formatting of a document as well as interchange. ODA incorporates formatting information directly into the document, through the specific layout structure and layout and presentation styles.

SGML has gained acceptance in the publishing industry, where the ability to either define specialized document type definitions and formatting conventions, or use those prepared by others, suits individual user-companies that are accustomed to making their own decisions about a document's appearance when rendered. ODA was originally designed with the office word processing environment in mind, assumes a more tightly coupled community of users, and provides for organizational standards for content and structure as well as appearance. Their resulting differences in syntax and philosophy notwithstanding, both standards are capable of expressing the essential structure and content of a document.

EXCHANGING DOCUMENTS BETWEEN THE STANDARDS

Communities of users have grown up around both SGML and ODA, along with tools and techniques peculiar to those communities. The standards themselves facilitate document interchange *within* these communities, but tools and techniques for the interchange of documents *between* these communities are not yet widely available or accepted. A tool that allows documents to be interchanged between SGML and ODA formats could then be regarded as a bridge, or gateway, between the two standards. Such a bridge should possess the following characteristics:

1. The bridge should be *reliable*, allowing for the migration of documents from one standard to the other with minimal (and ideally no) loss of information.
2. The bridge should be *automatic*, handling a wide variety of documents with minimal user intervention, but not at the expense of reliability.

The bridge may be used in two ways: for one-way translation of documents in which one standard is the source and the other is the target, and for two-way translations in which a document is moved from the source standard to the target with the intent of moving the document back to the source standard at some point in the future. A two-way translation can be viewed as the functional composition of two one-way translations. A reliable two-way translation must also preserve, or allow for the reconstruction of, any important information that might otherwise be lost as a result of either one-way translation.

A bridge between SGML and ODA needs to operate on a *structural* level as well as a *document* level. The SGML document type definition of a document contains structural information about a document (and, in fact, a class of documents) that should be preserved in the corresponding ODA document, whether as part of the specific logical structure of the ODA document, or as part of a generic logical structure that describes a set of ODA documents.

The content model of an element defined in an SGML DTD can most easily be captured in ODA as the value of the generator for subordinates attribute of a generic logical object. (Recall that the generator for subordinates attribute of an object specifies, in a

BNF-like notation, the types of objects that can appear as subordinates of that object.) The generator for subordinates attribute is optional, according to the ODA standard. If the generator for subordinates attribute is defined for each of the generic logical objects that comprise the generic logical structure of an ODA document, then those generator for subordinates attributes are said to make up a *complete generator set*.

The essence of the translation problem at the structural level is to determine whether the SGML DTD and the ODA generic logical structure, between which translation is to be done, are compatible. We say that an SGML DTD and an ODA GLS are *compatible* if and only if

1. each element in the SGML DTD corresponds to exactly one object class defined in the ODA GLS,
2. a complete generator set exists for the ODA generic logical structure, and
3. the generator for subordinates attribute of each generic composite logical object is equivalent to the content model of the corresponding SGML element.

Structural compatibility means that the content models in the SGML DTD and the complete generator set in the ODA generic logical structure are isomorphic, i.e. that the parse tree for an SGML document will be the same shape as the tree representation of the corresponding ODA document's specific logical structure. We make an exception for artificial basic logical objects like TEXT, whose only purpose is to connect a composite logical object like PARA to a content portion.

Structural compatibility reduces the translation problem at the document level to a relabeling of nodes in a tree, making it much easier to provide reliable automatic translation at the document level. The converse is also true: lack of structural compatibility makes it much harder to provide reliable automatic translation at the document level. If structural compatibility is lacking, however, all is not lost: a weaker notion of compatibility can be used for one-way translations on a document-by-document basis. A single ODA document is *sufficiently-compatible* with an SGML DTD if and only if

1. each object class referenced in the ODA document (excluding artificial objects like TEXT) corresponds to exactly one element in the SGML DTD, and
2. each specific logical object appearing in the ODA document has a subordinates attribute (possibly derived from the generator for subordinates attribute of a generic logical object) that conforms to the content model of the corresponding SGML element.

Similarly, an SGML-encoded document and an ODA generic logical structure are sufficiently-compatible if and only if

1. each element used in the SGML document corresponds to exactly one object class defined by an object in the ODA generic logical structure, and
2. the elements in the SGML document conform to the generator for subordinates attributes of the various generic logical objects.

If a document is sufficiently-compatible with a target structure, whether that target structure is an SGML DTD or an ODA GLS, translation of that particular document can still be done reliably and automatically. Several factors may interfere with structural compatibility:

- Structural compatibility is based on a one-to-one correspondence between the ODA document's logical object classes (excluding artificial objects) and the elements in the target SGML DTD. For example, such a correspondence exists between the elements in the SGML DTD shown in Figure 2 and the ODA generic logical structure shown in Figures 5 and 6. If no such correspondence exists, then significant user intervention may be needed to perform the conversions between tags and logical object classes.
- The SGML syntax for content models and the ODA syntax for the generator for subordinates attribute are similar, both being based on BNF. They differ, however, in detail. For example, there is no easy way in ODA to specify inclusions or exclusions.
- SGML's *concurrent markup* feature allows a document to contain two or more sets of elements. The DTD could define two or more sets of elements to describe the document's logical structure. The DTD could also define a set of tags that describe the intended layout of the document in addition to a set of tags that describe its logical structure. An SGML document with two sets of elements would be structurally compatible with ODA if one set of SGML elements corresponded to the ODA logical view and the other set corresponded to the layout view. ODA does not currently support multiple logical views, making it difficult to achieve structural compatibility between ODA and an SGML document with two or more sets of logical elements. (Concurrent markup is an optional feature of SGML that not all SGML parsers support.)

So far, we have assumed the existence of a "target" structure, whether it be an SGML DTD or an ODA GLS, to which we are trying to convert. Unfortunately, when mapping an ODA document to SGML, a target DTD may not be available. If the ODA generator for subordinates attributes are present in the ODA document, the values of these attributes can be used to synthesize the content models of a target DTD. The generator for subordinates attribute, however, is an optional attribute of generic logical objects, so there is a very real chance that the document may not have a complete set of them. If an ODA document does not have all the generator for subordinates attributes (i.e. it does not have a complete generator set), we might still be able to synthesize a target DTD by using the generator for subordinates attributes that are available, if any, and by seeing which objects occur as subordinates of other objects in a given document's specific logical structure. For example, if we notice that objects of class A, B and C appear as subordinates of objects of class X, then the objects A, B and C can then be used to define for object class X a "catchall" generator for subordinates attribute of the form "OPTREP(CHO(A, B, C)", which corresponds to the regular expression (and SGML content model) "(A | B | C)*" [10].

As we mentioned above, if structural compatibility exists, then document level translation involves a mapping between SGML elements and ODA logical object classes. Even with structural compatibility, however, an ODA document in formatted or formatted-processable form carries layout information that is not easily transmitted in SGML without the use of the optional concurrent markup feature.

We mentioned earlier that a bridge between SGML and ODA should be both reliable and automatic. Given structural compatibility (including an appropriate target), automatic conversion at the document level is feasible. SGML documents can be parsed, and

corresponding ODA objects generated, without human intervention. Similarly, it is not difficult to traverse the logical structure of an ODA document and generate corresponding SGML tags. The textual content in the ODA document will need to be scanned for special characters. For example, the character content architecture defines special characters to delimit subscripts and superscripts. Many SGML DTDs provide elements for the same purpose, and the mapping between these ODA special characters and the corresponding SGML tags would be straightforward. Non-textual content, such as raster graphics, can be placed in separate files and brought into the SGML document as entity references or with the NDATA feature.

A PROSPECTIVE BRIDGE: ODL

The ODA standard contains two formats that are meant to support document interchange. The first is the Office Document Interchange Format, or ODIF, which represents an ODA document as a binary encoding written in a general data structure description language known as ASN.1 (for Abstract Syntax Notation) [11]. The ASN.1 encoding is accompanied by an ASCII representation that is intended to be more easily read by humans. ASN.1 is described in detail in Tanenbaum [12].

The other interchange format is ODL, or Office Document Language [2]. ODL is an SGML application (i.e. a set of SGML document type definitions) that allows ODA documents to be represented in an SGML format. ODL is equivalent to ODIF in the sense that both are capable of expressing the structure and content of an ODA document in detail. ODIF uses ASN.1 to express this information. ODL allows SGML elements to be used to describe the ODA document.

In ODL, the user-visible name attribute of an ODA object is used to generate the corresponding SGML tag. In Figure 3, for example, each of the logical objects (except TEXT) has a user-visible name attribute corresponding to an element in the SGML DTD shown in Figure 2. As a result, when that ODA document is converted to ODL, the output is equivalent to the SGML document shown in Figure 1 (equivalent, but not identical, since the ODL document won't necessarily use shortened or omitted tags, and may add or delete white space between tags).

If the user-visible name attribute has not been supplied, ODL provides a set of elements that describe the various constituents, such as basic logical objects, pages, frames, and document layout root. Figure 7 shows a DTD for transmitting an ODA document's specific logical structure in the absence of user-visible names, along with the first few lines of an ODL document conforming to that DTD. The document logical root is marked by <dlor>, the composite logical objects such as PARA are marked by <clo>, and the basic logical objects are marked by <blo>. The <cp> tag denotes character content for a processable-form document. (Although ODA allows BLOs to have more than one subordinate content portion, ODL restricts each instance of a basic logical object to only a single subordinate) [2].

ODL allows ODA documents to be encoded in an SGML format. The ODL standard supplies DTDs for use with formatted, processable, and formatted-processable form documents. ODL was never intended to support translation of ODA documents to *arbitrary* SGML DTDs. When using ODL to translate an ODA source document to SGML, compatibility between the source document and the target DTD is the main concern. In

```
<!DOCTYPE dlor [
<!-- (C) International Organization for Standardization 1988
       Permission to copy in any form is granted for use with
       conforming SGML systems and applications as defined in
       ISO 8879, provided this notice is included in all copies.
-->
<!ELEMENT dlor  o o     (clo | blo)+    -- document logical root -->
<!ELEMENT clo   - -     (clo | blo)+    -- composite logical object -->
<!ELEMENT blo   - o     (cp)            -- basic logical object -->
<!ELEMENT cp    o o     (#PCDATA)       -- processable character content -->
<!ENTITY lt    "<" >
]>
<dlor><clo><blo><cp>
On the Interchangeability of SGML and ODA/ODIF
</cp></blo>
<clo><blo><cp>
Charles K. Nicholas
</cp></blo>
<blo><cp>
Lawrence A. Welsch
</cp></blo>
<blo><cp>
Office Systems Engineering Group
Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899
</cp></blo>
</clo>
<clo><clo><blo><cp>
SGML and ODA/ODIF are international standards for the markup and
interchange of electronic documents.  These standards are
incompatible, in the sense that a document encoded using SGML cannot
be used directly in an ODA-based system, and vice versa.  We
first describe these two standards, and suggest criteria under which a
bridge between the two standards could be evaluated.  We then evaluate
the Office Document Language (ODL), an SGML application specifically
designed for ODA/ODIF documents, with respect to these criteria.
</cp></blo>
</clo>
</clo>
</clo>
 .
 .
 .
```

Figure 7. A DTD for an ODA document's specific logical structure. In the absence of user-visible names, specific logical objects are named after their object class. (Adapted from ISO 8613, Part 5)

particular, if the target DTD and the ODA document are not structurally compatible, the user is responsible for bringing the resulting ODL document into conformance with the target DTD. For example, if an object in the ODA document belongs to an object class with the user-visible name "affiliation", and the target DTD had no corresponding element, then human assistance would be required to complete the translation.

Two problems make it difficult to perform reliable translations from ODA to SGML using ODL:

1. When translating an ODA document (or class of documents) to SGML, the GLS of the ODA document will contain valuable structural information.

Unfortunately, the ODL standard currently says very little about how a generic logical structure is to be represented.

2. When an ODA document lacking user-visible names is mapped to ODL, the object type is used to create the required ODL tags: basic logical objects are labeled with ODL's <blo> tag, composite logical objects are labeled with ODL's <clo> tag, and the document logical root is labeled with the <dlor> tag. Suppose, for example, that the user-visible name attributes had not been supplied for the objects in the ODA document shown in Figure 3. We would then have to use the <dlor>, <clo>, or <blo> tags in place of the <ARTICLE>, <ABSTRACT>, <PARA>, and other tags defined in Figure 2. This represents a significant loss of structural information.

A similar problem arises when translating from SGML to ODA: ODL makes no provision for encapsulating or otherwise preserving the structural information in the original DTD. No means of encapsulating the DTD of the original document is provided. Since ODL doesn't capture structural information, a translation system based solely on ODL must deal separately with the structural level issues of construction of a target DTD or GLS. This means that all sorts of structural information, including inclusions, exclusions, and concurrent markup, can be lost during translation from SGML to ODA via ODL. (Note that while ODA does not support concurrent documents in general, ODL does. In fact, concurrent markup is used in the definition of ODL to capture the logical and layout views of a document simultaneously.) Because ODL gives very little insight into preserving the structural information contained in generic logical structures, and makes no provision at all for preserving the structural information in SGML DTDs, ODL currently does not satisfy the reliability property.

If the documents being translated are either structurally compatible or sufficiently-compatible, one-way translation from ODA to SGML via ODL can be done reliably and automatically. The ODL version of an ODA document can be produced by traversing the ODA document and writing out the textual content and the user-visible names of objects, in appropriate ODL syntax, as each object is encountered. If there is (at least) sufficient compatibility between the ODA document and the target DTD, each object can be mapped into a corresponding element in the target DTD. If no target DTD exists, one can be synthesized by using the information in the generic logical structure's generator for subordinates attributes, or the specific logical structure's subordinates attributes. Otherwise, human assistance may be needed to map composite and basic logical objects to the correct elements in a target DTD.

One-way translation from SGML to ODA via ODL is slightly more complicated. If we assume sufficient compatibility between the SGML document and the target generic logical structure, then each SGML tag in the source document corresponds to an object class defined by a logical object in the target GLS. Then a given element name can be used verbatim in the corresponding ODL tag, which in turn can be used to compute the object class and user-visible name of the corresponding ODA object. Character data within tags can be mapped to character content inside a basic logical object. For example, if an <ABSTRACT> element appears in the SGML source document, that tag can be written as is into the generated ODL document. When the ODL stream is converted to ODA, the object class defined by the generic logical object with the user-visible name "ABSTRACT" can be identified, and an ODA logical object (such as PARA) containing the text of the abstract can be attached as a subordinate.

In summary, the main obstacles to reliable translation between ODA and SGML via ODL are the loss of structural information in the DTD or GLS, and the loss of layout information in formatted and formatted-processable form ODA documents. Given that one-way translation between a compatible source and target is relatively easy to automate, two-way translations, which can be thought of as successive one-way translations, are also easy to automate.

DESIGN OF ODL-BASED TRANSLATORS

We have written two programs that demonstrate the interoperability of ODA and SGML, using the ideas presented in this paper. These programs were designed to perform two-way translations between ODA and SGML documents conforming to the DTD in Figure 2. SGML_to_ODA converts SGML documents into ODA. ODA_to_SGML reads an ODA document from an ODIF file and converts that document to ODL, generating start and end tags from each object's user-visible name.

Converting from SGML to ODA

In two-way translation, the structural information in the source DTD must be preserved and transmitted with the document when it is sent to ODA, so that sufficient compatibility between the new ODA document and the original DTD can be maintained. The translation program can preserve structural information by converting the content models of the SGML elements into generator for subordinates attributes of corresponding ODA generic logical objects, thereby creating a target generic logical structure. Another option is to save the original DTD, and use "catchall" generator for subordinates attributes in the target generic logical structure.

Once the target GLS is ready, the SGML-encoded document must be parsed. Four factors should be considered during the parsing process:

- Entity references that resolve to strings of characters may appear in SGML documents, and these references should be preserved in the character content portions of basic logical objects during conversion from SGML to ODA. These same entity references will then be present if the ODA document is later translated to ODL. (It is for this reason that the entity "lt" defined in the original DTD in Figure 2 also appears in Figure 7.)
- Entity references may also refer to local files containing non-character data, such as raster graphics or line drawings. The parser would need to copy these files into content portions conforming to an appropriate content architecture.
- Attributes may occur within SGML tags. The DTD describes the attributes that may be associated with specific elements. For example, a figure reference element may have the form <FIGREF ID=n>, where the value of the ID attribute is the number of the figure being referenced. SGML attributes can be simulated using the ODA bindings mechanism, which allows a name/value pair to be associated with a specific or generic object. When an attribute declaration appears in the definition of an SGML element, a binding can be defined for the corresponding generic logical object. A default value can be associated with the binding at the time of definition, and instances of that attribute value can be

associated with corresponding specific logical objects as the attribute is encountered in SGML tags.

- No construct in ODA corresponds to the notion of marked sections in SGML. According to Goldfarb, "A marked section of a document is one that has been identified for a special purpose, such as ignoring markup within it" [6]. One appropriate use of the marked section feature, in this context of document interchange, would be to select portions of an SGML document that are not to be converted to ODA.

Given that these caveats are observed, then the translation routine can parse the SGML document, copying content at the leaves of the parse tree into ODA content objects, and mapping elements into basic and composite logical objects. If software to create ODA documents is available (the ODA Toolkit built as part of the EXPRES project provides this functionality) [13], then the translation routine may build the desired ODA document directly. Otherwise, the output of the translation program would be an ODL or ODIF data stream.

Converting from ODA to SGML

When designing a program to convert a document from ODA to SGML, the main problem is to guarantee structural compatibility. Recall that ODA documents can be in formatted, processable, or formatted-processable form. Since the lack of structural information in a formatted form document makes structural compatibility hard to attain, we restrict ourselves to documents in processable or formatted-processable form. We first consider the translation of processable form documents, and then discuss the extra steps needed to prevent loss of layout information during the translation of formatted-processable form documents.

If structural information is lacking, structural compatibility becomes harder to guarantee. There are four cases to consider:

1. If the ODA document's GLS and a target DTD are both available, we can test their structural compatibility by building a one-to-one correspondence between ODA object classes and SGML tags. If the GLS and the target DTD enjoy structural compatibility, then we can produce an ODL version of the ODA document, generating tags in the ODL document that conform to the target DTD.
2. If the GLS is available but not the target DTD, the generator for subordinates attributes in the generic logical structure can be used to synthesize an appropriate target DTD.
3. If the target DTD is available but not the GLS, then human assistance may be needed to figure out which tags in the target DTD correspond to the various basic and composite logical objects in the ODA document.
4. If neither the target DTD nor the GLS are available, then we can generate an ODL file using the <dlor>, <clo>, and <blo> tags.

If no generic logical structure is available, then translation from ODA to SGML is more difficult. If we were willing to settle for partial automation, it would be relatively

easy to build a general tool for mapping ODL tags to some other SGML DTD that would occasionally need to ask a human user which of several candidate tags (in the target DTD) is appropriate. If the target DTD had a regular structure, e.g. of the form title, author, sequence of paragraphs, then it would be easy to implement a translation routine as a finite state machine, for example. However, such a routine would not respond well (e.g. by generating tags that don't conform to the target DTD) to ODL documents that deviate from the pattern. A somewhat more specialized pattern-matching program (based, for example, on attribute grammars rather than finite state machines) would require more effort to build, but would be able to process conforming documents with relatively little human intervention. (This was the approach used in the Chameleon document translation system) [14].

When ODL is used to encode a formatted-processable document, containing both logical and layout information, each content portion needs to be tagged in accordance with what are, in effect, two different SGML DTDs: One for describing logical structure, and the other for describing layout structure. If the concurrent markup feature is not available, one alternative approach is to use a single document encoded with logical tags, and generate a version of the same document with layout tags via the SGML link mechanism. The link mechanism performs naive tag substitution, however, and cannot make decisions based, for example, on the nesting level at which a given tag resides. The link mechanism requires *a priori* knowledge of how each tag is to be laid out, without regard to nesting level, or the values of SGML or ODA attributes [3].

Implementation

SGML_to_ODA begins by reading a table listing the elements in the SGML document and the object class identifiers to be used for the corresponding ODA generic logical objects. This is used to create an ODA document that contains a structurally compatible generic logical structure with user-visible names, but which is otherwise empty. SGML_to_ODA then reads and parses an SGML-encoded document. During the parse, the content and SGML tags in the SGML document are converted to ODA objects in the specific logical structure. Each basic and composite logical object is assigned an object class identifier corresponding to a basic or composite logical object in the generic structure. SGML_to_ODA then creates an ODIF version of the ODA document, including the generic and specific logical structures, that can be saved or transmitted as needed. The current version of SGML_to_ODA allows entity references to be present in the SGML document, but makes no attempt to resolve them.

SGML_to_ODA was written in YACC, Lex and C. Public-domain versions of YACC and Lex were used to generate a small, DTD-specific SGML parser. The action routines in the parser make use of library calls in the ODA Toolkit [13] to build the various components of the ODA document. We considered using the NBS SGML parser [15], but that parser does not provide for action routines.[2]

ODA_to_SGML begins by reading an ODIF file from disk. ODA Toolkit calls are made to convert the ODIF file into an internal ODA data structure. The specific logical structure of this ODA document is then traversed, left to right, in preorder. As the ODA

[2] We did use the NBS SGML parser, as well as Clark's version of *arcsgml* for UNIX, to verify the correctness of the example document and the DTDs shown in Figures 2 and 7.

document is traversed, the character content and user-visible names of the object classes appearing in the specific logical structure are written as an ODL output file.

These two programs were tested on an early version of this paper, marked-up in accordance with the DTD in Figure 2. The SGML input file was 45 701 bytes (982 lines) in length. Twenty-eight SGML elements were used in the document, and several elements were set up to allow start-tags or end-tags to be omitted. Otherwise, no effort was made to minimize markup in the test document. When reading the SGML version of this paper and writing the corresponding ODIF document, SGML_to_ODA ran at approximately 70 lines per second on a lightly loaded SUN 4/280. The resulting ODIF file was 100 624 bytes in length. The size increase can be explained by noting that the ODIF file contained descriptions of twenty-nine generic logical objects (corresponding to the original SGML elements plus the TEXT basic logical object), and a number of ODA attributes, such as object identifier, that are implicit in the SGML document.

The ODA document was then converted back to SGML with ODA_to_SGML. Start-tags and end-tags were generated in full, even if they had been omitted or abbreviated in the original document, except for TEXT objects, which were written as character data with no surrounding <TEXT> tags. The other differences between the resulting SGML document and the original were due to SGML_to_ODA's treatment of whitespace. During translation to ODA, some newline characters that occurred between tags in the original document, in locations where text was permitted, were interpreted as text and were placed in the ODA document as character content. Whitespace that occurred between tags in places where text was not allowed was ignored. No human intervention was required during either translation.

CONCLUSION

We have described the SGML and ODA standards, and developed a set of criteria for evaluating a bridge mechanism between SGML and ODA. ODL's weakness as a bridge comes from its lack of a mechanism to preserve the structural information in an SGML DTD or an ODA generic logical structure. However, if the documents being exchanged are structurally or sufficiently compatible with the target standard, then reliable automatic translation can be achieved. We have developed software that performs such translation.

When planning to convert documents from one standard to the other, it is important to make sure that the source document(s) and the target standard are compatible. SGML features that can interfere with structural compatibility, such as inclusions, exclusions, and concurrent markup, should be used with care. ODA documents should be represented in processable or formatted-processable form, so that explicit structural information in the form of generic logical objects or user-visible names can be preserved. If ODL is to be used for interchange, then the structural information in the source document, in the form of an SGML document type definition or an ODA generic logical structure, should be maintained.

We have shown that a translator based on the idea of structural compatibility can operate reliably and automatically. In addition, our experience suggests that under some circumstances SGML may be preferable to ODIF as an interchange format for ODA documents, due to the compactness of SGML files in comparison to ODIF, and the fact that SGML files can be read by humans and ODIF files cannot.

We expect ODA and SGML to coexist in the typical office environment. ODA's ability to handle logical and layout components of a document should make ODA compatibility an important feature of document processing systems in the future. However, the tasks of authoring and formatting/composing are frequently separated in practice. This is the case, for example, in traditional publishing. At the moment, the larger number of SGML-compatible systems will cause SGML to remain the standard of choice in such situations, but this may change as the number of ODA-compatible systems increases.

ACKNOWLEDGEMENTS

We extend our thanks to John Barkley, Frank Dawson, Fran Nielsen, Sandy Mamrak, Judi Moline, Mark Sherman, and the anonymous referees for their comments on earlier versions of this paper.

REFERENCES

1. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, Ref. No. ISO 8879-1986(E). First edition — 1986-10-15. FIPS PUB 152. October 1986.
2. *Information Processing — Text and Office Systems — Office Document Architecture (ODA) and Interchange Format*, ISO 8613 Parts 1–8. 1989.
3. Martin Bryan, *SGML: an author's guide to the Standard Generalized Markup Language*, Addison-Wesley, Reading, MA, 1988.
4. Association of American Publishers, *Reference Manual on Electronic Manuscript Preparation and Markup*, EPSIG c/o OCLC, Dublin, OH 43017-0702, 1989.
5. *Technical Manuals: Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text*, U.S. Department of Defense, July 1990.
6. Charles F. Goldfarb, *The SGML Handbook*, Oxford University Press, Oxford, 1991.
7. Wolfgang Horak, 'Office Document Architecture and Office Document Interchange Formats: current status of international standardization', *IEEE Computer*, **18** (10), 50–60 (1985).
8. R. Hunter, P. Kaijser, and F. Nielsen, 'ODA: a document architecture for open systems', *Computer Communications*, **12** (2), 69–79 (1989).
9. *Final Text, ISO/IEC CD 10179, Information Technology — Text and Office Systems — Document Style Semantics and Specification Language (DSSSL)*, 1991.
10. Golkar, Seyed N., *Experience with Document Interchange*, IBM/NSF Workshop on Compound Document Exchange using ODA, 1989.
11. *Information Processing — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1)*, ISO 8824 1987.
12. Tanenbaum, Andrew S., *Computer Networks*, Prentice Hall, Englewood Cliffs, NJ, 1988.
13. Jonathan Rosenberg, Mark Sherman, Ann Marks, and Jaap Akkerhuis, *Multi-media Document Translation: ODA and the EXPRES Project*, Springer-Verlag, New York, 1991.
14. Mamrak, S., Kaelbling, M., Nicholas, C., and Share, M., 'Chameleon: a system for solving the data translation problem', *IEEE Transactions on Software Engineering*, **15** (9), 408–414 (1989).
15. Jim Heath and Larry Welsch, 'Difficulties in Parsing SGML', in *Proceedings of the ACM Conference on Document Processing Systems*, ACM Press, Santa Fe, New Mexico, pp. 71–77, December, 1988.

APPENDIX

The following listing shows a portion of the ODA document that resulted from translating the SGML example document to ODA. Each constituent of the ODA document is

shown with its required attributes. As each constituent was created, the ODA Toolkit assigned it a number of the form *document_number/constituent_number*. Because the SGML document was parsed in a bottom-up fashion, the content portions and basic objects were created before their parent objects. Therefore, most objects have a *higher* constituent number than their subordinates. The ODA document shown belongs to the processable document class, so no layout view exists.

We made two simplifying assumptions with respect to the generator for subordinates attributes: we allowed any composite logical object to be a subordinate of the document logical root, and we allowed any composite or basic logical object to be a subordinate of any composite logical object. This field is therefore rather lengthy, and we omit it from this listing for the sake of brevity. We assigned object class identifiers to the various generic logical objects in sequence, according to the order in which those objects appeared in Figures 5 and .

The generic composite logical objects SUBSEC, SSECHEAD, EMTEXT, FIGURE, CODE, GRAPHIC, CAPTION, ENDNOTES, APPENDIX, and APPHEAD have been omitted from this listing, since none of those objects were mentioned (via their SGML counterparts) in Figure 1. We added line breaks in the content information fields of several content portions for the sake of readability.

```
Generic Logical Structure:

1/1: GENERIC DOCUMENT LOGICAL ROOT
    ("Object Class Identifier" "2")
    ("Object Type" 'document logical root')
    ("User-visible Name" "ARTICLE")

1/2: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 0")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "FRONTM")

1/3: GENERIC BASIC LOGICAL OBJECT
    ("Object Class Identifier" "2 0 0")
    ("Object Type" 'basic logical object')
    ("User-visible Name" "TITLE")

1/4: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 0 1")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "AUGROUP")

1/5: GENERIC BASIC LOGICAL OBJECT
    ("Object Class Identifier" "2 0 1 0")
    ("Object Type" 'basic logical object')
    ("User-visible Name" "AUTHOR")
```

```
1/6: GENERIC BASIC LOGICAL OBJECT
    ("Object Class Identifier" "2 0 1 1")
    ("Object Type" 'basic logical object')
    ("User-visible Name" "AUAFF")

1/7: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 0 2")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "ABSTRACT")

1/8: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 1")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "BODY")

1/9: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 1 0")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "SEC")

1/10: GENERIC BASIC LOGICAL OBJECT
    ("Object Class Identifier" "2 1 0 0")
    ("Object Type" 'basic logical object')
    ("User-visible Name" "SECHEAD")

1/13: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 1 0 2")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "PARA")

1/14: GENERIC BASIC LOGICAL OBJECT
    ("Object Class Identifier" "2 1 0 2 0")
    ("Object Type" 'basic logical object')
    ("User-visible Name" "TEXT")

1/21: GENERIC COMPOSITE LOGICAL OBJECT
    ("Object Class Identifier" "2 1 0 2 6")
    ("Object Type" 'composite logical object')
    ("User-visible Name" "FN")

1/22: GENERIC BASIC LOGICAL OBJECT
    ("Object Class Identifier" "2 1 0 2 7")
    ("Object Type" 'basic logical object')
    ("User-visible Name" "ENR")
```

```
Specific Logical Structure:

1/30: SPECIFIC DOCUMENT LOGICAL ROOT
    ("Object Class" 1/1:)
    ("Object Type" 'document logical root')
    ("Subordinates" 1/44: 1/762: 1/773: 1/779:)
    ("User-visible Name" "ARTICLE")

1/44: SPECIFIC COMPOSITE LOGICAL OBJECT
    ("Object Class" 1/2:)
    ("Object Type" 'composite logical object')
    ("Subordinates" 1/32: 1/39: 1/43:)
    ("User-visible Name" "FRONTM")

1/32: SPECIFIC BASIC LOGICAL OBJECT
    ("Content Architecture Class" 2 8 2 6 1
        ['processable character content architecture'])
    ("Content Portions" 1/31:)
    ("Object Class" 1/3:)
    ("Object Type" 'basic logical object')
    ("User-visible Name" "TITLE")

1/31: CONTENT PORTION (character)
    ("Content Information" ' 12On the Interchangeability of SGML
        and ODA/ODIF 12')
    ("Type of Coding" 2 8 3 6 0
        ['character content encoding'])

1/39: SPECIFIC COMPOSITE LOGICAL OBJECT
    ("Object Class" 1/4:)
    ("Object Type" 'composite logical object')
    ("Subordinates" 1/34: 1/36: 1/38:)
    ("User-visible Name" "AUGROUP")

1/34: SPECIFIC BASIC LOGICAL OBJECT
    ("Content Architecture Class" 2 8 2 6 1
        ['processable character content architecture'])
    ("Content Portions" 1/33:)
    ("Object Class" 1/5:)
    ("Object Type" 'basic logical object')
    ("User-visible Name" "AUTHOR")

1/33: CONTENT PORTION (character)
    ("Content Information" ' 12Charles K. Nicholas 12')
    ("Type of Coding" 2 8 3 6 0
        ['character content encoding'])
```

```
1/36: SPECIFIC BASIC LOGICAL OBJECT
    ("Content Architecture Class" 2 8 2 6 1
        ['processable character content architecture'])
    ("Content Portions" 1/35:)
    ("Object Class" 1/5:)
    ("Object Type" 'basic logical object')
    ("User-visible Name" "AUTHOR")

1/35: CONTENT PORTION (character)
    ("Content Information" ' 12Lawrence A. Welsch 12')
    ("Type of Coding" 2 8 3 6 0
        ['character content encoding'])

1/38: SPECIFIC BASIC LOGICAL OBJECT
    ("Content Architecture Class" 2 8 2 6 1
        ['processable character content architecture'])
    ("Content Portions" 1/37:)
    ("Object Class" 1/6:)
    ("Object Type" 'basic logical object')
    ("User-visible Name" "AUAFF")

1/37: CONTENT PORTION (character)
    ("Content Information" ' 12Office Systems Engineering
Group 12Computer Systems Laboratory 12National Institute of
Standards and Technology 12Gaithersburg, MD 20899 12')
    ("Type of Coding" 2 8 3 6 0
        ['character content encoding'])

1/43: SPECIFIC COMPOSITE LOGICAL OBJECT
    ("Object Class" 1/7:)
    ("Object Type" 'composite logical object')
    ("Subordinates" 1/42:)
    ("User-visible Name" "ABSTRACT")

1/42: SPECIFIC COMPOSITE LOGICAL OBJECT
    ("Object Class" 1/13:)
    ("Object Type" 'composite logical object')
    ("Subordinates" 1/41:)
    ("User-visible Name" "PARA")

1/41: SPECIFIC BASIC LOGICAL OBJECT
    ("Content Architecture Class" 2 8 2 6 1
        ['processable character content architecture'])
    ("Content Portions" 1/40:)
    ("Object Class" 1/14:)
    ("Object Type" 'basic logical object')
    ("User-visible Name" "TEXT")
```

```
1/40: CONTENT PORTION (character)
    ("Content Information" ‘ 12SGML and ODA/ODIF are international
standards for the markup and 12interchange of electronic documents.
These standards are 12incompatible, in the sense that a document
encoded using SGML cannot 12be used directly in an ODA-based system,
and vice versa. ‘ ...)
    ("Type of Coding" 2 8 3 6 0
        [’character content encoding’])
```