CMU-ITC-081
EUUG Conference Proceedings
September 1989
Vienna

Processable Multimedia Document Interchange using ODA[1]

Jaap Akkerhuis
Ann Marks
Jonathan Rosenberg
Mark S. Sherman

Information Technology Center
Carnegie Mellon University
Pittsburgh, USA
jaap+@andrew.cmu.edu
annm+@andrew.cmu.edu
jr+@andrew.cmu.edu
mss+@andrew.cmu.edu

ABSTRACT

The EXPRES (Experimental Research in Electronic Submission) project promotes the electronic interchange of multi-media documents among the scientific research community. For this project we concentrate on the problem of effective interchange of processable multi-media documents. In particular, we are ignoring the transfer method. Instead we concern ourselves with the question of how a multi-media document created on one system can be viewed and edited on another system.

The obvious technique of performing translations between each pair of systems is impractical. In order to attack the problems efficiently, we make use of a standard representation. We have settled on the international standard Office Document Architecture (ODA) [ISO88a] as the intermediate format. This paper discusses how we implemented ODA for interchange.

Introduction

In the last decade there has been an explosion in the number of multimedia document processing systems. These range from simple batch text processors systems to fancy WYSIWYG multimedia editors; these are used by professional typesetters, computing professionals and administrative personnel. Most systems have "multimedia" facilities, which range from the ability to use different fonts, inclusion of drawings, and mathematical equations, up to sound and video. The term "multimedia document" is most of the time actually a misnomer. Basic facilities such as non-proportional spacing, different fonts, etc. have always been part of a document. We will use this term however to discriminate from the typewriter and lineprinter style documents.

The introduction of these systems has generated a new problem. To interchange a document from one system to the other is hardly possible given the number of systems in use.

The National Science Foundation (NSF) receives yearly a considerable number of proposals for research grants. Most of these are actually prepared on the above mentioned systems. This observation leads to the question of whether it would be possible to receive these proposals in electronic form, so that the amount a paper involved could decrease. Ideally, the electronic form should make it possible to process the documents, without requiring the submitters to standardise on a single system. This led to start of the Experimental Research in Electronic Submission (EXPRES) project, whose goal is to investigate the problems of document interchange for dissimilar systems. Main participants are the Information

[1] This work was funded by the US National Science Foundation under grant ASC-8617695. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the US Government.

Technology (ITC) Center at Carnegie Mellon University (CMU) and the Center for Technology Integration (CITI) at the University of Michigan (UM).

As part of the Andrew project [Mor86a], the ITC developed a multimedia toolkit, the Andrew Toolkit (ATK) [Pal88a], which supports objects such as multi-font text, line drawings, equations, spread sheets, and raster drawings. At the University of Michigan, the Diamond multimedia system [Tho85a] is the base for their collaboration in the project. Although Diamond and the Andrew systems are quite similar in their capabilities, they were independently developed and quite different in their underlining implementations. This provided an ideal environment for the EXPRES project.

1. Translation fidelity

When translating a document from one system to be viewed and edited on another, it is necessary to decided on the fidelity required. We distinguish three major fidelity categories for document interchange. These are *imagining fidelity, structural fidelity* and *editing fidelity*. Below we briefly discuss what these entail. An more thorough discussion may be found in [Ros89a].

1.1. Imaging fidelity

Imaging fidelity can defined by how close the translated document matches the original in appearance when printed or on the screen. For certain types of document this is a primary requirement, as for instance legal documents where changes in the layout are often unacceptable. This type of fidelity is also what naive users want from a translation system.

To achieve this, one can use a standard page description language, such as PostScript, Interpress or DVI, as produced by TeX. Of course this assumes that the implementation of these language on the receiving end will produce the the same result. This might not always be the case. For example, if the more or less standard Computer Modern Roman fonts for TeX are not used, this scheme will fail.

1.2. Structural fidelity

Normally a document is highly structured. A document consists of paragraphs, figures with legends, footnotes etc. Maintaining the structure of the document allows the receiver to format the document differently than the originator. This way one can retain the general appearance of an document.

1.3. Editing fidelity

Editing fidelity requires structural fidelity, but, in addition, the document must be editable in a way similar to the originating system. This is particularly important for the EXPRES project since we are concerned with allowing collaboration on multimedia documents from dissimilar systems.

A prime example of an editing feature to be retained during translation is *style sheet* or *property sheet* information. A lot of document systems provide a mechanism for defining styles. For example, one can define a quotation style where the right and left margin are indented and the font changed to italic. This style can then be applied to various parts of the text. The important fact is that when this style's definition is changed, this will take effect on the parts of the text where the style is applied. To elaborate, let's assume that we have a document on system A, and that the document includes a definition for a quotation style which is applied several places in the document. Let's assume that we now interchange the document to system B, and that the interchange preserves editing fidelity. On system B, an edit is made to change the quotation style. Now all applications of the quotation style will appear different in the document on system B. Of course it must be possible to interchange these changes to the quotation style when sending the document back to the original system or to any other system.

2. The Office Documentation Architecture

It is obvious that it is impractical to build $n \times (n-1)$ translators for n document processing systems. We decided to translate in and out a common format for all translators. For the this format we choose the Office Documentation Standard (ODA), an international standard designed for interchange of multimedia documents. One of the main reasons is that it doesn't only specify the logical structure of a document but also includes full semantics to specify the layout of a document. We felt that this was necessary in the interchance, since users would insist on the ability to specify the appearance of the document. In addition, ODA is an international standard that has a following in Europe. This offered us the possibility of extending our interchange to others.

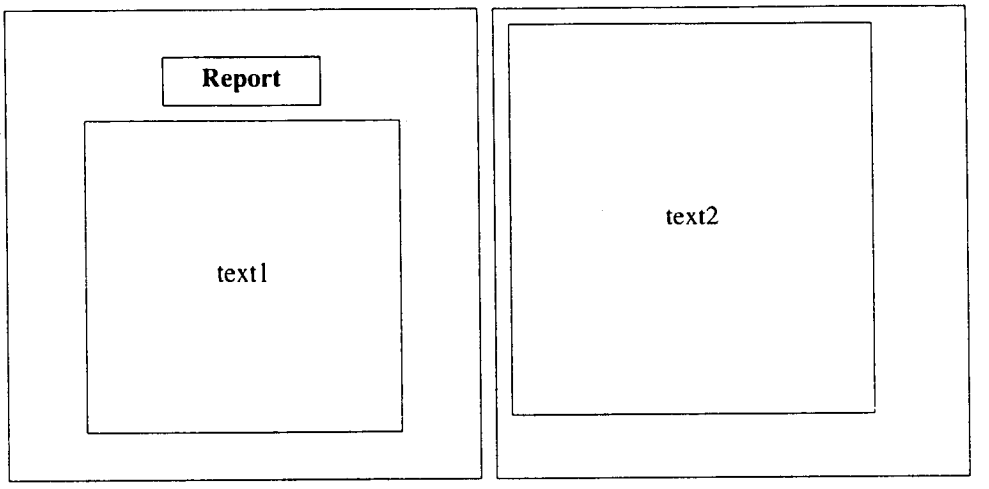

Figure 1: *The sample document*

ODA defines a document architecture, several content architectures and two data stream formats. The *document architecture* is the means by which the structure of a document, independent of its actual content, is represented. In general, an ODA document is represented using two sets of structures.† The *logical structure* is based on the meaning of various divisions of the document. For example, the logical structure of a document might consist of chapters, sections and paragraphs. In the *layout structure*, the document is structured on the basis of presentation. For example, the layout structure of a document might consist of pages and, within the pages, frames and blocks that define headers, footers and paragraphs.

In addition, each structure may exist in two forms: *generic* and *specific*. A generic structure may be thought of as a template or macro that allows structure information to be collected and referenced. For example, the *generic logical structure* of a document might indicate that the document consists of a title, followed by one or more sections, followed by a set of references. Correspondingly, a *generic layout structure* for the same document might indicate that the title is a block that appears two inches from the top of the first page and is centered.

If the generic structures of a document can be thought of as macros, then the specific structures represent invocations of those macros. The *specific logical structure* is, thus, the actual structure of a document. For example, the specific logical structure might show that a particular document consists of a title, five sections and a set of seven references. There is a *specific layout structure*, corresponding to the generic layout structure, but it is used only for the representation of a final form document (one that may be imaged). Since we are concerned only with editable documents, our translation schemes do not use any specific layout structures. The actual content of an ODA document consists of instances of *content architectures*. Each content architecture defines its own internal structure, which may consist of logical and layout structures. There are currently three content architectures defined within ODA. *Character content architecture* defines the presentation and processing of characters and allows the specification of graphic character sets, multiple fonts, ligatures and formatting directives such as indentation and justification. *Raster graphics content architecture* defines pictorial information represented by an array of picture elements. *Geometric graphics content architecture* defines picture description information such as arcs and lines.

A *data stream* is an out-of-memory representation for a document that is suitable for storage in a file or transmission over a network. The ODA standard defines an ASN.1 binary data stream format known as the Office Document Interchange Format (ODIF).‡

† It is possible for an ODA document to consist of only one of these sets of structures. For our purposes, this is immaterial and we will only consider documents containing both sets of structures.

‡ ODA defines another data stream representation, the Office Document Language (ODL), which is a clear text representation that conforms to the Standard Generalized Markup Language standard (SGML). Note that this does not imply that there is a direct relationship between an ODA document and the equivalent document marked up using SGML.

Documents represented in ODA are graphs, the nodes of which are known as *constituents*. Each constituent has a set of *attribute-value* pairs. The values of attributes are used to represent the structure of the document. Attributes have values that control the presentation and layout of the document. For example, the value of the attribute "Separation" at a constituent will control the distance between blocks of text when the document is displayed or imaged.

Figure 1 displays a small example of a two page document. It consists of two pages. The first page contains a title "**report**" that is centered and bold-faced, and a centered paragraph, "text1". The second page contains a left justified paragraph "text2".

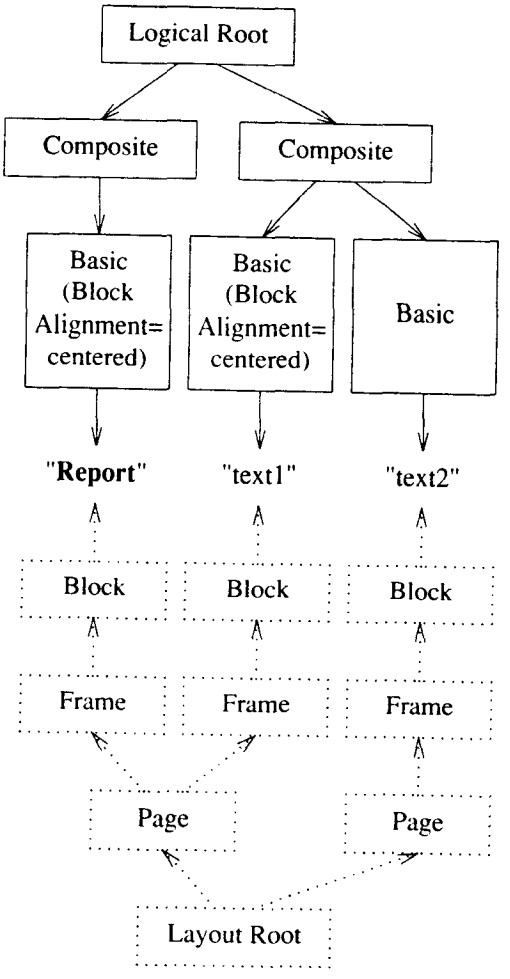

Figure 2: *The ODA specific structure*

The corresponding specific logical and layout structures are shown in Figure 2. The logical structure consists of a *composite* object for each section and a *basic* object for each paragraph. The centering of the text is accomplished by attaching the attribute "Block Alignment" with the value "centered" to two of the basic objects. The specific layout structure consists of two *page* objects each of which contains a *frame* and a *block* object to position the paragraph.

3. The CMU ODA Tool Kit

To make it easier to build translators we developed a subroutine library for manipulating ODA documents. The Tool Kit [Rosa], written in C, includes C language definitions for the objects that occur in ODA, such as constituents and sequences. The Tool Kit also includes data type definitions of all the data used in ODA, for example construction expressions and font information. The Tool Kit provides subroutines for manipulating ODA structures. For example, the Tool Kit permits the creation of documents and components; it allows the user to associate an attribute value with a constituent; subroutines for reading and writting the binary ODIF interchange format are also included.

The Tool Kit performs a number of useful functions for dealing with ODA. When setting an attribute value, the Tool Kit performs full semantic checking to ensure that the attribute can be associated with the given constituent, and that the value specified for the attribute is legal. This is extremely useful because some attributes are only allowed on certain kinds of constituents, so the Tool Kit will prevent the creation of illegal combinations. The ODA standard also specifies a complicated scheme for defaulting of attribute values. These defaulting rules are fully supported by the Tool Kit. The Tool Kit includes subroutines for reading and writing of the ODIF data stream, operations which are complex given that the ODIF stream is a context sensitive binary representation. For debugging purposes it is possible to create a human readable representation of the binary data stream. Service routines are also included to support the ISO 9541 standard for fonts which is used by ODA.

The subroutine library is designed to be highly portable, therefore it is written in a subset of C which we carefully specified with portability in mind. The Tool Kit will compile and run on various operating systems and hardware platforms. By carefully separating the operating system dependencies, such as I/O, into separate modules, it is easy to port to other systems. In addition, hardware dependencies are localised to a short set of type definitions. Currently there is support for:

1 UNIX (System V and BSD flavors) on Vaxes, IBM-RT, Sun

2 VAX VMS

3 Macintosh under MPW

4 IBM PC running MS-DOS

We estimate that it is less than a day's work to bring it up on a new machine.

Although the Tool Kit is very useful, there are many capabilities that it does not currently include. For example, there is no capability for interpreting content. The actual content of the document, be it text or a raster, is a sequence of bytes. Translator implementors must examine content sequences to extract formating information such as font changes. The Tool Kit does not include any of the conventional document notions such a paragraph or left margin. Such higher level document constructs must be built by creating the appropriate ODA structure and attaching the required attributes. There are some document operations that would make nice additions to the Tool Kit. For example, it would be convenient to be able to have a single operation for instantiating a generic object when constructing an ODA document. Some of the information required by ODA is very cumbersome, e.g. construction expressions and font information. It would be nice to be able to specify these concisely. The Tool Kit does not perform the layout or imaging processes included in ODA's document reference model. Both could be built on top of the Tool Kit. Finally, there is no support for the ODL SGML based interchange format.†

4. Examples of Tool Kit Use

In this section, we present four examples illustrating the use of the Tool Kit to construct translators. These four examples are paired to show similar processing operations when translating from a native document format to ODA and when translating from ODA back into a native format. The first pair of examples shows how a part of the specific logical structure is created or examined. In the second pair, we show how to associate attribute values with constituents or how to retrieve attribute values. Throughout, we use a C-like notation to present program segments. We omit checking of Tool Kit return values to keep the examples uncluttered.

4.1. Example of Document Structure

In this example, we assume that the document being interchanged is to contain processable information. In ODA, this is represented using the logical structure. We further limit ourselves to the specific logical structure to illustrate how the structure is built up when translating to ODA, and how the structure is interpreted when translating from ODA. Figure 3 depicts the ODA structure. Here we assume that there is a parent component that is a composite logical object. The children of this parent are also to be composite logical objects.

† Although the binary ODIF representation of a document is cryptic and unreadable by a human, it is also much easier to parse and unparse than ODL. In addition, all other ODA implementations of which we were aware were using ODIF and we would, thus, have some chance of interchanging with other systems. For these reasons, we have only implemented reading and writing of ODIF within the ODA tool Kit.

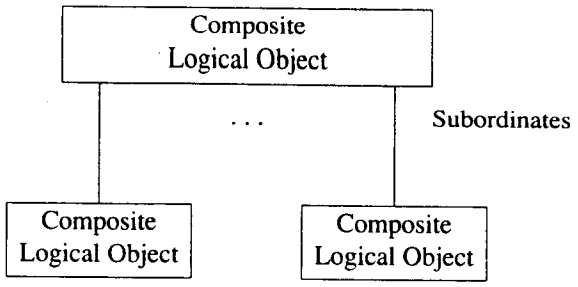

Figure 3: *ODA Structure for Examples 1 and 2*

```
/*
 * At this point, a parent composite logical object is
 *  to be created.
 * The children, also composite logical objects, are also
 *  to be created.
 * The subordinates attributes is to connect the parent to
 *  the children.
 * document is assumed to be of DOCUMENT_type and created by
 *  a call to the Tool Kit MakeDocument routine.
 */

INT_type ReturnCode;     /* for Tool Kit returns */
CONSTITUENT parent;      /* the parent */
CONSTITUENT child;       /* one of the children */
SEQUENCE_CONSTITUENT_type Subordinates;
                         /* the value for the subordinates attribute */

    /* create the parent */
    Parent = MakeComponent( document, SPECIFIC_COMPONENT,
               at_OBJECT_TYPE_comp_logical_obj );

    /* create the empty sequence for the subordinates attribute */
    Subordinates = MakeSequence( SEQUENCE_CONSTITUENT_tag, (INT_type) 0 );

    /* loop to create the children and add to the subordinates */
    for( each child needed ){
        /* make the child component */
        Child = MakeComponent( document, SPECIFIC_COMPONENT,
                  at_OBJECT_TYPE_comp_logical_obj );

        /* expand the subordinates sequence */
        ReturnCode = ExpandSequence( Subordinates, (INT_type) 1 );

        /* add the Child at the end of Subordinates */
        Subordinates->sequence_value.constituents[Subordinate->length-1]
                                                              = Child;
    }

    /* now set the subordinates attribute */
    ReturnCode = SetAttr( parent, at_SUBORDINATES,
                    (POINTER_type) Subordinates,
                    (PARM_MASK_type) 0 );
```

Example 1: *Creating Components and adding the Subordinates Attribute*

4.1.1. Translating into ODA

To translate into ODA, the native document must be traversed, and the appropriate ODA structures constructed. We assume in this example that the traversal has reached a point where the parent component is

to be built, the children are to be created, and the subordinates attribute is to be associated with the parent. The code for accomplishing this is given in Example 1.

In this example, we see calls to the Tool Kit MakeComponent routine which creates a component. MakeComponent creates the component in the given document: the component will be of a given type, a specific component in this example; the component will be of a given kind, a composite logical object in this example. The Tool Kit MakeSequence routine is used to create a sequence of constituents to hold the value of the subordinates attribute. Initially, this sequence has length zero, but the length is increased by one as each child is created. Finally, the SetAttr routine is used to set the value of the parent's subordinate attribute. Because the subordinates attribute does not have parameters (as does the offset attribute, for example), a null value is passed as the last parameter.

4.1.2. Translating out of ODA

When translating out of ODA, the ODIF data stream will first be read from a file by calling the ReadODIF Tool Kit routine. This will result in the creation of a document with type DOCUMENT_type. This example shows how the data stream is read, and how to examine the children encountered during the traversal. As the traversal is performed, the native form of the document is constructed; we omit code for doing this. The code is shown in Example 2.

In this example, we see that the entire ODIF data stream is read by a single call to the Tool Kit routine ReadODIF. This creates a DOCUMENT_type object that is the document contained in the ODIF data stream. To locate the root of the specific logical structure, the FindDocumentRoot is called. At this point the recursive traversal begins. To traverse each constituent in the specific logical structure, each constituent is processed; this processing will entail the creation of the appropriate native document format piece. To continue the traversal, the value of each constituent's subordinate attribute is obtained, and traverse is called for each subordinate.

The Tool Kit provides an alternate way to traverse document structures using the Tool Kit ITERATOR_type object. In Example 3, we outline how an ITERATOR_type can be used for this purpose.

Note that this example begins like the previous one with the reading of the data stream and the locating of the document's specific logical root. The iterator is then created, and the iteration is performed. This iteration will result in the entire specific logical structure being traversed. Note that the traversal will be parent first, like the previous example, but, here the traversal is breadth first where the previous example is depth first. Finally, this example illustrates an iterative way to traverse document structure in contrast to the previous example which used recursion.

4.2. Example including an Object Class and a Style

The next pair of examples is based on the ODA structure shown in Figure 4. Here we have three constituents: a basic logical object, a basic logical object class and a presentation style. The basic logical object indicates that it is an instance of the basic logical object class by the object class attribute. The basic logical object class has an associated presentation style as indicated by the presentation style attribute. The presentation style has one attribute associated with it, the character content architecture attribute indentation. The indentation attribute has value 5 which, according to ODA semantics, is in standard measurement units.

4.2.1. Translating into ODA

This example illustrates how the structure shown in Figure 4 can be created. The native format document is being traversed. At some point, in this traversal the structure shown in Figure 4 needs to be created to represent the native format document. The code for doing this is shown in Example 4.

Note that each constituent is created by a Tool Kit call. The two components are created using MakeComponent, but the presentation style must be created using MakeStyle. The attribute values are set using various flavors of the SetAttr routine. To set the object class attribute for the basic logical object, and to set the presentation style attribute for the basic logical object class, the SetAttr routine is used. To set the value of the indentation attribute on the presentation style, we have used the SetIntAttr routine. This permits us to pass the value of the attribute rather than the address of an INT_type variable with value 5 which SetAttr would require.

```
/*
 * A data stream is read.
 *
 * The document specific logical root is located.
 *
 * A depth first, parent first traversal is performed on
 *  the specific logical structure.
 */

INT_type ReturnCode;        /* Tool Kit return value */
DOCUMENT_type document;     /* the document */
CONSTITUENT LogicalRoot;    /* the root of the specific logical structure */

    /* first read in the document */
    document = ReadODIF( fileno(stdin) );
    /*
     * Here we assume that this is running on UNIX
     *  and that the data stream is on the standard input.
     */
    /* now locate the document logical root */
    LogicalRoot = FindDocumentRoot( document, SPECIFIC_DOC_LOGICAL_ROOT );

    /* call subroutine traverse to examine the root */
    traverse( LogicalRoot );

void traverse( constituent )
CONSTITUENT constituent;
{
    /*
     * Traverse the given constituent.
     *
     * The appropriate part of the native format would
     *  be created but this is not shown.
     */

    INT_type i;      /* for looping through the children */
                     /* the constituent's subordinates */
    SEQUENCE_CONSTITUENT_type Subordinates;
    INT_type ReturnCode;     /* return code from the Tool Kit */

    process the parent;

    /* now get the parent's subordinates attribute */
    ReturnCode = GetAttr( constituent,
                    at_SUBORDINATES,
                    (POINTER_type) &Subordinates,
                    BOOL_false,    /* do not use the ODA defaulting rules */
                    (PARM_MASK_type *) 0 ) ;

    /* now start the iteration over the children */
    for( i = (INT_type) 0; i < Subordinates->length; i++ ){
            /* recursively traverse the child */
        traverse( Subordinates->sequence_value.constituents[i] );
    }
```

Example 2: *Reading a Data Stream and Traversing the Document*

4.2.2. Translating Out Of ODA

To translate out of ODA, the ODA document is traversed. Presumably, at some point the basic logical object is encountered, and the value of the indentation attribute is needed. Example 5 shows how to obtain the value of the indentation attribute.

```
DOCUMENT_type document;          /* the document */
ITERATOR_type iterator;          /* the iterator */
CONSTITUENT constituent;         /* a constituent */

    /* first read in the document */
    document = ReadODIF( fileno(stdin) );
    /*
     * Here we assume that this is running on UNIX
     *  and that the data stream is on the standard input.
     */

    /* now locate the document logical root */
    constituent = FindDocumentRoot( document, SPECIFIC_DOC_LOGICAL_ROOT );

    /* make the iterator */
    iterator = MakeSubgraphIterator( constituent,
                PARENTS_FIRST,    /* the parent goes before the children */
                BREADTH_FIRST );  /* the traversal is to be breadth first */

    /* now begin the iteration */
    for( constituent = NextConstituent( iterator );
            constituent != ERROR_CONSTITUENT
                        && constituent != NULL_CONSTITUENT;
            constituent = NextConstituent( iterator ) ){

                process constituent;
    }
```

Example 3: *Reading a Data Stream and Traversing the Document using an Iterator*

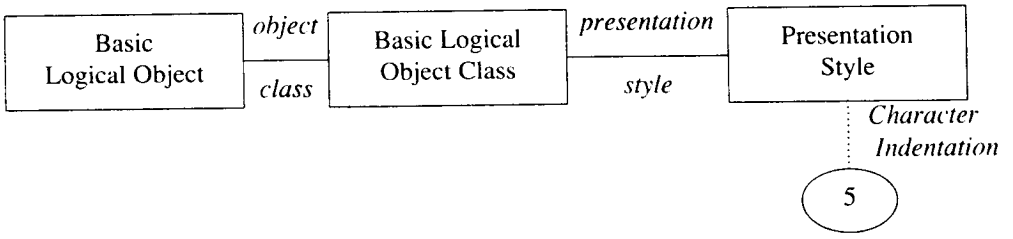

Figure 4: *ODA Structure for Examples 4 and 5*

We see that only one Tool Kit routine is called, GetAttr. The value of indentation for the BasicLogicalObject can be obtained easily by using the ODA defaulting mechanism which is implemented by the Tool Kit. Without Tool Kit support for ODA defaulting, it would be necessary to look for styles, object classes, resource documents, default value lists, document application profile defaults, and to know the ISO 8613 default values for all attributes that have default values.

5. Conclusions and Status of the CMU ODA Tool Kit

The ODA Tool Kit enabled us to interchange documents between different platforms and between different document processing systems in timely fashion. Having the ODA Tool Kit as a common base permitted us to interchange much sooner than we would outwise have been able done otherwise. In addition, many difficulties were eliminated because we were all using the same Tool Kit.

We plan to release the Tool Kit on the next MIT X tape, which is currently scheduled for release in December 1989, although MIT is controlling the date of the release. We are also investigating the possibility of releasing the Tool Kit through other publically available channels, possible the ISODE distribution. On release the Tool Kit will be largely complete. The functionality that we presently expect to be missing or limited includes: limited support for ODIF, most notably the document profile and specific layout structure will be largely incomplete or missing; the Tool Kit will include no support for swapping of

```
/*
 * First create the constituents, then set the appropriate
 *  attributes.
 *
 * document is a DOCUMENT_type object created by a
 *  call to the Tool Kit routine MakeDocument.
 */
CONSTITUENT BasicLogicalObject;
CONSTITUENT BasicLogicalObjectClass;
CONSTITUENT PresentationStyle;
INT_type ReturnCode;    /* Tool Kit return code */

    /* make the basic logical object */
    BasicLogicalObject = MakeComponent( document,
                SPECIFIC_COMPONENT,    /* in the specific structure */
                at_OBJECT_TYPE_bas_logical_obj );

    /* make the basic logical object class */
    BasicLogicalObjectClass = MakeComponent( document,
                GENERIC_COMPONENT,     /* in the generic structure */
                at_OBJECT_TYPE_bas_logical_obj );

    /* make the presentation style */
    PresentationStyle = MakeStyle( document,
                PRESENTATION_STYLE );

    /* now associate the basic logical object with the object class */
    ReturnCode = SetAttr( BasicLogicalObject,
            at_OBJECT_CLASS,    /* the attribute to be set */
            (POINTER_type) BasicLogicalObjectClass,
                                /* the value of the attribute */
            (PARM_MASK_type) 0 );
                /* the object class attribute does not have parameters */

    /* now associate the basic logical object class with the style */
    ReturnCode = SetAttr( BasicLogicalObjectClass,
            at_PRESENTATION_STYLE,    /* the attribute to be set */
            (POINTER_type) PresentationStyle,
                                /* the value of the attribute */
            (PARM_MASK_type) 0 );
                /* the presentation style attribute does not have parameters */

    /* now add the indentation style value to the style */
    ReturnCode = SetIntAttr( PresentationStyle,
            cc_INDENTATION,     /* the attribute to be set */
            (INT_type) 5,       /* the value of the attribute */
            (PARM_MASK_type) 0 );
                /* the indentation attribute does not have parameters */
```

Example 4: *Building the ODA Structure shown in Figure 4*

parts of the ODA document, a feature important for machines with limited memory, or when working with huge documents; no ability to evaluate the expressions included in ODA, e.g. string expressions, numeric expressions etc.; the Tool Kit only supports text and raster content. At present, the Tool Kit is about 80,000 lines of C.

References

[ISO88a] ISO, *Office Document Architecture (ODA) and Interchange Format (ISO 8613)*, International Organization for Standardization (ISO), 1988.

[Mor86a] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal, and F. Donelson Smith, "Andrew: A Distributed Personal Computing Environment," *Communications of the ACM*, vol. 29, no. 3, pp. 184–201, March 1986.

```
/*
 * BasicLogicalObject is the basic logical object as
 *  shown in Figure 4.
 */
INT_type ReturnCode;     /* Tool Kit return value */
INT_type Indentation;    /* value of indentation */

    ReturnCode = GetAttr( BasicLogicalObject,
                          /* the attribute whose value is sought */
            cc_INDENTATION,
                          /* where to return the value of indentation */
            (POINTER_type) &Indentation,
            BOOL_true,    /* use the ODA defaulting rules */
                          /* do not return the parm mask */
            (PARM_MASK_type *) 0 );
```

Example 5: *Extracting the Value of Indentation for the BasicLogicalObject*

[Pal88a] Andrew. J. Palay, Wilfred J. Hansen, Mark Sherman, Maria G. Wadlow, Thomas P. Neuendorffer, Zalman Stern, Miles Bader, and Thom Peters, "The Andrew Toolkit—An Overview," *Proceedings of the USENIX Winter Conference*, pp. 9–21, USENIX Association, Berkeley, CA, February, 1988.

[Ros89a] Jonathan Rosenberg, Mark S. Sherman, Ann Marks, and Frank Giuffrida, "Translating Among Processable Multi-media Document Formats Using ODA," *Proceedings of the ACN Conferenrence on Document Processing Systems*, pp. 61–70, ACM, New York, December 5-9, 1989.

[Rosa] Jonathan Rosenberg, Ann Marks, Mark Sherman, Paul Crumley, and Maria Wadlow, "The CMU ODA Tool Kit: Site Installation Guide & Application Programmer's Interface," Technical Report CMU-ITC-071, Information Technology Center, Carnegie Mellon.

[Tho85a] Robert H. Thomas, Harry C. Forsdick, Terrence R. Crowley, Richard W. Schaaf, Raymond S. Tomlinson, and Virginia M. Travers, "Diamond: A Multimedia Message System Build on a Distributed Architecture," *IEEE Computer*, vol. 18, no. 12, pp. 65–78, December 1985.

```
/*
 * BasicLogicalObject is the basic logical object as
 *  shown in Figure 4.
 */
INT_type ReturnCode;     /* Tool Kit return value */
INT_type Indentation;    /* value of indentation */

    ReturnCode = GetAttr( BasicLogicalObject,
                          /* the attribute whose value is sought */
            cc_INDENTATION,
                          /* where to return the value of indentation */
            (POINTER_type) &Indentation,
            BOOL_true,    /* use the ODA defaulting rules */
                          /* do not return the parm mask */
            (PARM_MASK_type *) 0 );
```

Example 5: *Extracting the Value of Indentation for the BasicLogicalObject*

[Pal88a] Andrew. J. Palay, Wilfred J. Hansen, Mark Sherman, Maria G. Wadlow, Thomas P. Neuendorffer, Zalman Stern, Miles Bader, and Thom Peters, "The Andrew Toolkit—An Overview," *Proceedings of the USENIX Winter Conference*, pp. 9–21, USENIX Association, Berkeley, CA, February, 1988.

[Ros89a] Jonathan Rosenberg, Mark S. Sherman, Ann Marks, and Frank Giuffrida, "Translating Among Processable Multi-media Document Formats Using ODA," *Proceedings of the ACN Conferenrence on Document Processing Systems*, pp. 61–70, ACM, New York, December 5-9, 1989.

[Rosa] Jonathan Rosenberg, Ann Marks, Mark Sherman, Paul Crumley, and Maria Wadlow, "The CMU ODA Tool Kit: Site Installation Guide & Application Programmer's Interface," Technical Report CMU-ITC-071, Information Technology Center, Carnegie Mellon.

[Tho85a] Robert H. Thomas, Harry C. Forsdick, Terrence R. Crowley, Richard W. Schaaf, Raymond S. Tomlinson, and Virginia M. Travers, "Diamond: A Multimedia Message System Build on a Distributed Architecture," *IEEE Computer*, vol. 18, no. 12, pp. 65–78, December 1985.