Building a Document Management System:

The Issue of Interoperability

Brendan Hills and Anne-Marie Vercoustre
Melbourne Laboratory
CSIRO Division of Information Technology
723 Swanston Street, Carlton,
Victoria, 3053, Australia

Brendan.Hills@dit.csiro.au
Anne-Marie.Vercoustre@dit.csiro.au

Abstract

Interoperability between software tools is an important issue in document computing, especially when interchange formats such as SGML are considered. This paper provides a discussion of our attempts to address the issue of interoperability between third party SGML software tools. The context for this is our current project SIMDAS (SIM Document Authoring System) which aims to integrate third party SGML editing tools with the SIM SGML database as a first step to form a complete document management system.

Keywords:
Document authoring, Document management, Document computing, HTML, Interoperability, Open systems, SGML, Word.

1. Introduction

The Structured General Mark-up Language (SGML) [ISO:2] is an attempt to provide interoperability between disparate document tools, thereby increasing the longevity of document data and simplifying the process of exchanging documents between systems. These issues have long been important to large organisations such as the Department of Defence and government departments and libraries - accordingly, much of the support for SGML has come from these quarters. Increasingly, large commercial organisations, particularly those that deal with these government entities, have become interested in (or even forced to use) SGML.

A much more recent development in SGML has been its use in the World Wide Web (WWW). One of the WWW's primary goals is to be platform independent, and this is where SGML has been very useful in defining the Hyper Text Mark-up Language (HTML) the language of documents on the WWW. It is the use of HTML in the WWW which has given SGML its greatest exposure, which has sparked an increased interest in building tools such as databases, editors and document management systems which work with SGML.

HTML and SGML both provide interoperability at the document level which allows different tools to work with the same documents. This is just one of several levels of interoperability which are provided by different systems. At a low level, the existence and success of the Internet is made possible by standards such as TCP/IP, FTP, and (more recently) HTTP, which provide network level interoperability. At a higher level, standards such as CORBA [OMG] provide platform independence by specifying a mechanism by which software objects can communicate with each other either on a single machine or across a network. Two implementations of this sort of technology are OpenDoc [OPENDOC] Microsoft's OLE[1] [OLE]. These systems enable applications to exchange data and to send message to each other (implemented by the sender calling a function defined by the recipient). They have both been aimed in particular at the idea of "compound documents" in which a document becomes a conglomerate of different types of data, each which is handled by an appropriate software object. [ORFA96]

This issue of software interoperability will be important for any software system which attempts to integrate with, extend on, or interact with third party software systems. Interoperability is particularly relevant to document management systems for two reasons. Firstly, document management - taken to be

[1] When OLE was first developed by Microsoft it was an abbreviation of "Object Linking and Embedding", but OLE now does a lot more than is implied by that term so the full name is no longer used - see [BROC95] for a detailed discussion of the development of OLE.

Proceedings of SGML Asia-Pacific'96,
Sydney Australia,
Speteember 24-26 1996.

"the automated control of electronic documents...throughout their entire life cycle" [CLE95] - is inherently multifaceted in its requirements. It will therefore tend to suit solutions which combine a number of tools, rather than a monolithic application. Secondly, document standards such as SGML give us a format which enables different software tools to work on the same document.

Systems such as OLE and OpenDoc or IBM's "BluePrint" [BLUE] are very useful in providing a mechanism for software to interoperate when that software is designed with interoperability in mind. What we have found however, is that many of the tools in which we are interested have not been designed to provide this software level of interoperability. It is this finding which has lead us to develop the system for third party tool integration which this paper describes.

Our project is to provide a Windows based document authoring system for Ferntree and the Royal Melbourne Institute of Technology's (RMIT) "Structured Information Manager" (SIM) SGML document database system [RMIT95:1]. This document authoring system is aimed to create a complete document management system when included with the SIM database. It includes tools for editing SGML documents, displaying SGML document on the screen, representing hyperlinks graphically and providing editing facilities for those links.

There are currently many SGML editors available - commercially and otherwise - and although hyperlink management tools are not yet common (however see "Navipress" for a good example [NAV]), it is likely that they will become increasingly so, partially due to the popularity of hypertext on the WWW. Our intention is to provide software level interoperability between existing tools, rather than to write our own, and also to enable the system to incorporate new tools as they are developed. This has the advantage of providing a flexible authoring system in terms of both capacity and price, and of significantly reducing the scope of our project. A major part of the project then has been to provide operability between "third party" software tools and SIM. These tools have been found to vary considerably in their "openness" from being completely closed to having available source code, and use a variety of communications protocols such as OLE and DDE.

This issue of software level interoperability will be important for any software system which attempts to integrate with, extend on, or interact with other third party software systems. Interoperability is particularly relevant to document management systems for two reasons. We define document management to be "the automated control of electronic documents...throughout their entire life cycle".[CLE95] Firstly, document management is inherently multifaceted in its requirements. This means that it will tend to suit solutions which combine a number of tools rather than a monolithic application, as described by the Document Management Alliance.[DMA] Secondly, document standards such as SGML give us a format which allows us to work on the same document with different programs.

This paper provides background on the current project - SIMDAS - and on the SIM database and the SIM PC client - SIMPCI [RMIT95:2]. It then describes the requirements of the current project. A brief examination is presented of possible architectures for SIMDAS - with SIMDAS integrated tightly with the SIMPCI; and with SIMDAS dependant on, but separate to the SIMPCI.

Following this is a detailed discussion of issues for SIMDAS which have arisen from its requirement to incorporate third party software tools. An overview and examples of our DEScript integration scripting language are given - including the integration of Microsoft Word. This is followed by further description of the SIMDAS approach to integration a varying levels of openness.

2. Background

2.1 SGML - Document Level Interoperability

SGML (Standard Generalised Mark-up Language - ISO Standard 8879) is a language with which we can write a specification for the structure of a type of document - a Document Type Definition (DTD). There are two main advantages of using SGML, that the documents are structured and that they conform to a standard.

Using documents that are structured provides us with the means to use the document differently for different applications, different users, or on different occasions. This comes about because different software packages can choose to show particular parts of a document structure and they are not constrained to format the document in any particular way, rather they are able to display the document as the viewer requires.

The advantage of using a standard for document structure definition is that it ensures that documents are able to be used by different pieces of software, different versions of software, software from different manufacturers, old and new systems etc.

To make the most of these two features of SGML documents - that of structure and of standardisation - it would be ideal if we were able to transfer documents directly from one software package to another and even better, to be able to connect different pieces of document computing software together to perform the tasks that we require.

The major goal in our SIMDAS project was just this - to connect the "Structured Information Manager" SGML database (SIM), to a document authoring system (DAS). In this paper we examine the outcomes of this project provide specific examples of the general problem of the interoperability of third party tools.

2.2 The Structured Information Manager (SIM)

SIM is a multi-gigabyte document database management system and full-text information retrieval engine designed and engineered by the RMIT Multimedia Database group and marketed and sold by the Ferntree Computer Corporation. SIM has a client-server based architecture based on the ANSI Z39.50 [ISO: 1] client-server protocol with servers running on Sun SPARC workstations running the Solaris operating system and clients running on other machines which have software which supports the Z39.50 protocol.

The SIM search and retrieval engine is based around an SGML parser, giving SIM the power to index large collection of structured documents by their structural elements (e.g. Author, Title or References) as well as by their text. SIM maintains a complete inverted index of the document text including words which are normally excluded by stop lists such as "and" and "the".

SIM supports a range of user interface platforms, including a command line interface for ASCII terminals, a graphical user interface for a PC running Microsoft Windows (SIMPCI), and a World Wide Web client for browsers such as Mosaic and Netscape.

2.3 The SIM PC Interface (SIMPCI)

The SIMPCI is a SIM database query client which runs on a PC running the Microsoft Windows operating system (3.x, NT or 95). It allows the user to perform all of the searches which are supported by SIM including basic keyword searches, ranked searches, searches on structural elements of documents etc. In addition, the SIMPCI enables a user to perform standard Boolean operations on the result sets that are returned by these search operations and to save results sets.

Documents which have been returned in the results sets may be viewed using the "LivePage", from Inforium Inc.[INFO], which renders the SGML document on the screen according to a user defined style file.

2.4 The Document Authoring System

The SIMPCI provides the PC/Windows user with a mechanism to perform queries on a SIM database and to display the results of those queries. The SIMDAS project is to extend this functionality on the Windows platform by providing a authoring system which will enable the PC/Windows user to author SIM documents. The term "authoring" is used here to refer to the process of both creating new documents and editing existing ones. It includes authoring hyperlinks, graphics, other included documents, and DTD's as well as the text of the document itself. The document authoring systems could consist of either a single tool or a set of tools which provide different functions - e.g. A hyperlink editor, a version control tool and a SGML editor. The complete system may therefore form a document management system if so desired, or simply provide editing facilities for a library-like information service.

3. The SIMDAS Project

The aim of SIMDAS project (SIM Document Authoring System) is to provide a document authoring system for users on the PC to author documents for a SIM database. At the time that the prototype was commenced, the broad objectives were to build an authoring system for the SIM database that is:

- PC Based (Windows 3.x)[2]
- Can be sold cheaply, perhaps around $100 AUD per user.
- An optional add-on to SIMPCI - to be sold separately, as required.
- Separate from SIMPCI and SIM to avoid intellectual property issues between the two groups.

A more detailed set of requirements for the SIMDAS project was developed with an emphasis on generality so as not to limit the implementation options. The requirements for the SIMDAS project were as follows:

[2] This was subsequently changed to be Windows NT or Windows 95 as it became clear that these operating systems would replace Windows 3.1 and that they offer considerable adavntages.

- Search and browse documents in a SIM database.
- Load documents from a SIM database and edit them on-line as well as off-line. Here, “On-line” editing means users can edit documents while stay connecting to the server. “Off-line” editing means users can save the documents locally, disconnect from the server, and edit the documents at a later stage.
- Create new documents and store them in a SIM server.
- Create hypertext links.
- Follow existing hypertext links.
- Integration with other SGML editors
- Support document formats other than SGML
- Integration with other SGML database systems
- Handle large documents which are fragmented in the SIM database
- View and edit multimedia objects such as pictures.

It was decided that since there were already many good SGML editors available on the market, both as share-ware and commercial product, we would not attempt to build our own editor, rather we would build SIMDAS around an existing SGML editor.

3.1 The Prototype - Integrate SIMPCI with an SGML Editor.

The prototype of SIMDAS was built between October 1994 and March 1995 with a view to exhibiting it at the CeBIT conference in Hannover. We selected the "TagWizard" SGML editor to be the basis of SIMDAS as it is a relatively inexpensive product and because it is an add-on to the highly popular Windows based word processor "Microsoft Word". TagWizard adds SGML editing capabilities to Word, for example: ensuring that tags are inserted only where they are legal, whilst providing the user with a WYSIWYG view of their documents.

Figure 1 shows the architecture of the SIMDAS prototype. As can be seen, SIMDAS consists of a communications "Driver" which manages the DDE (Dynamic Data Exchange) communication between the SIMPCI and the TagWizard editor. In this figure, the dotted rectangle which delineates the scope of SIMDAS project is shown to overlap the SIMPCI and the TagWizard editor to indicate that modifications of both packages were required in order to provide the required DDE communications.

During the development of the prototype, we were given access to the source code of the TagWizard macros by NICE Technologies (though not their SGML parser, nor the internals of their main libraries) and we were therefore able to provide a smooth integration between SIM and TagWizard.

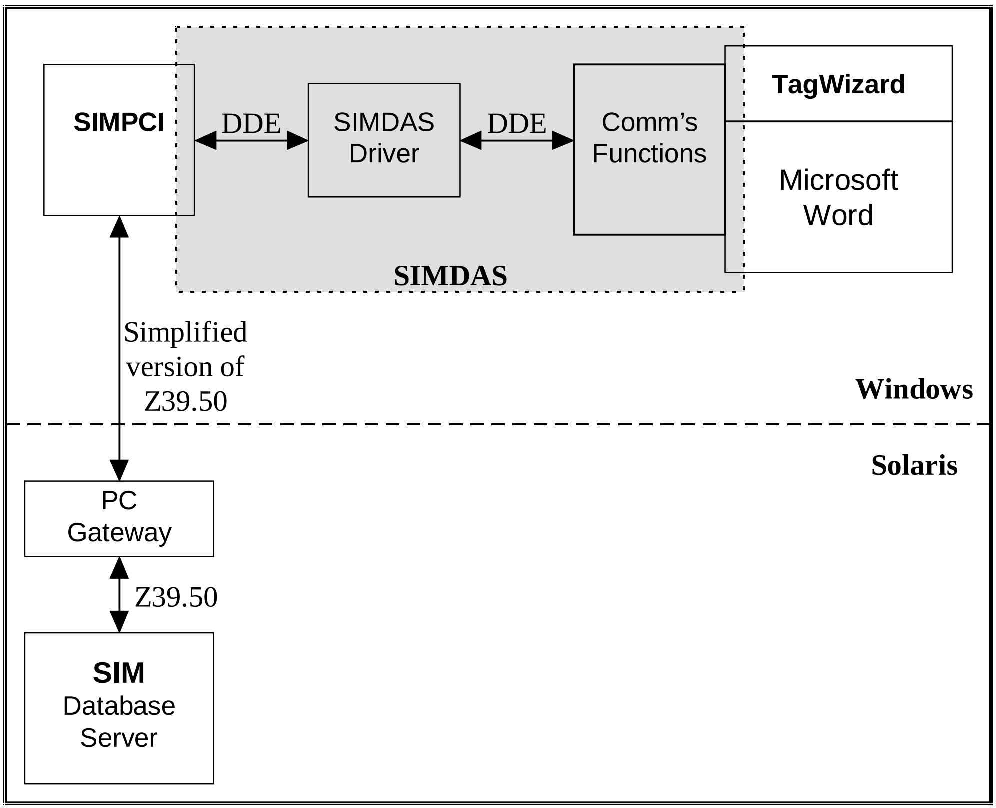

Figure 1 - SIMDAS Prototype Architecture

3.2 Discussion

The SIMDAS prototype was successful in that it enabled us to prove the concept of connecting an SGML editor to SIMPCI. It enabled a user to retrieve a document from a SIM database using the SIMPCI's search facilities and to then make changes to that document using the TagWizard SGML editor and to store those changes back to the SIM database where they were immediately available to all users.

Unfortunately though, there were several problems with the SIMDAS prototype which prompted us to rethink the design for the next version (Version 1). The first problem was in the choice of SGML editors as we found that the version of TagWizard which we used (admittedly and old one) suffered from problems of speed and reliability.

The other major problem with the SIMDAS prototype was the limitation of being locked into using a single SGML editor. As previously discussed, the very nature of an SGML document is that it may be used by a range of different types of users and it may contain many different kinds of information, ranging from single WWW pages to enormous CALS documents. This variability means that it is highly likely that different users will require different kinds of editing tools for their SGML documents, and that TagWizard will not suit every one.

On top of these two problems is the obvious issue of maintainability that we had incurred by modifying the source code of the TagWizard SGML editor

4. Version 1 - Integrate with Closed Editors

Version 1 of SIMDAS was started in April 1995 and was designed to provide a more robust implementation of SIMDAS and to extend on the prototype by overcoming the limitations discussed in section 3.2. During the design of the first version of SIMDAS, two different approaches were considered as follows.

Integrate with Several Editors using Public Interface
The most immediate problem with the SIMDAS prototype was the difficulty in adapting SIMDAS to upgrades in the TagWizard SGML editor.

The obvious solution to this problem was to try an integrate with TagWizard at level of a public interface - using Microsoft's OLE protocol for example. However TagWizard does not have this capacity so we were not able to use this approach with that editor - although perhaps we could use it with a different one.

In order to give the SIMDAS user greater flexibility in their choice of SGML editor we investigated the possibility of integrating with a variety of editors. This was considered in the light of the point about maintainability made above - as trying remain compatible several different editors would be much more difficult than with just a single editor. With this aim, we surveyed twelve SGML editors and nine HTML editors (See Appendices A and B) and examined their public interfaces to work out how we could use a generic mechanism for integrating with several of them, rather than writing customised code for each one.

We discovered a significant problem with this approach however. We found that despite the recent attention on "standards" such as OLE or OpenDoc, very few of the SGML editors on the market actually had public interfaces and when they did have public interfaces, they tended to differ greatly from each other. We found that the SGML editors on the market varied in their degree of "openness" as follows:

Completely closed
1. Command line options
2. Modifiable configuration file
3. Macro language
4. (Source code is available)
5. API is available
6. DDE communication
7. OLE communication
8. CORBA communication

Most editors fell into categories 1 to 3 - either completely closed, with some command line options, or with configuration files. This made our task of generalising communication with an editor at the level of the public interface very difficult, if not impossible.

Integrate with Closed Editors

The problems with the above approach lead us to consider a second possibility - that we would try to integrate with completely closed editors without modifying their source code. This solution enables us to avoid the problem of a lack of consistent public interfaces by not relying on them at all. An extension of this idea is to make integration configurable by the user so that as new products become available or the users needs change, they can adapt SIMDAS to suit.

4.1 Integrating with Closed Editors

Integrating with completely closed editors essentially means taking the "lowest common denominator" and saying "if we can integrate with a closed editor, we can integrate with <u>any</u> editor". The approach we took to this kind of integration was to look at the surveyed editors to determine what kinds of things SIMDAS would need to do to integrate with a closed editor. We then designed SIMDAS to have two separate components - the Document Exchange Manager (DXM), which can perform all the required functions for the integration between an editor and the SIMPCI, and a set of scripts, written in a purpose built script language "DEScript (Document Exchange Script), which instructs the DXM what is required to integrate with a particular editor. This script based approach is commonly used in distributed computing and according to [ORFA96] "[i]ncreasingly, scripts will become the predominant paradigm for orchestrating intercomponent collaboration..." (p. 560).

The architecture of Version 1 of SIMDAS is shown in Figure 2

4.2 Document Exchange Manager

The DXM performs four main sets of tasks. The first is to handle the communication with the SIMPCI. We elected to use the OLE protocol for this connection as it is stable, flexible and fast and required us to make far fewer alterations to the SIMPCI source code than the previous DDE protocol did. Microsoft is also phasing out use of the DDE protocol in favour of OLE and OLE is now well supported in development tools such as Visual C++ [VISC++] and Visual Basic [VISBAS], so the transition seemed sensible from this perspective.

The second major task of the DXM is to interpret the script which tells it how to integrate with a particular editor. The script consists of instructions about what files are needed and when they should be put, what command line arguments are used and the user interaction which is required to load and store[3] a file using the given editor. The DXM interprets these instructions and carried our the functions that they describe.

The third task of the DXM is to track the support files that the editor requires. The term "support files" refers to files such as style files, DTD's and SGML declaration files which all editors require. These files are stored in the SIM database and a retrieved by the DXM when the editors requires them and cached on the PC's hard disk so that they are readily available for subsequent edit sessions.

Finally, the DXM provides a simple interface which enables the user to select the editor which is to be used from a list of those editors which have been in-

[3] The term "Store" as opposed to "Save" is used throught out this discussion to distinguish the act of storing a document to the database from saving it to the PC's hard disk.

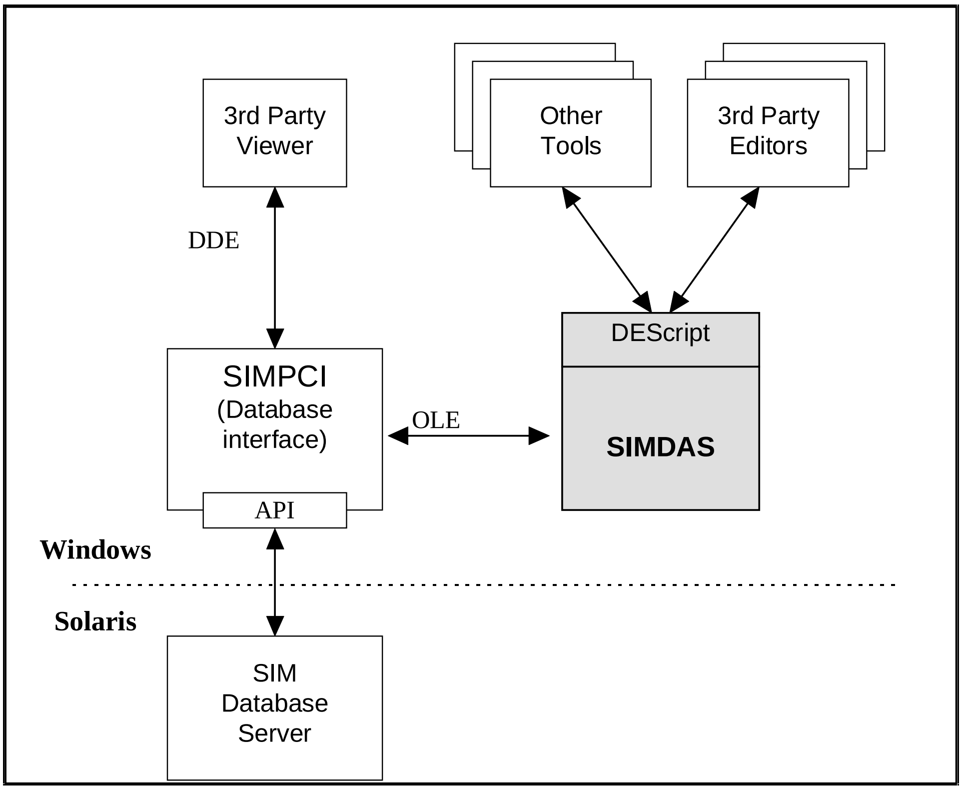

Figure 2 - SIMDAS Version 1 Architecture

tegrated, or to write the scripts to integrate a new editor with SIMDAS.

4.3 Configuration Scripts

Approach

SIMDAS configuration scripts are written in a purpose-built script language called "DEScript" (Document Exchange Script). This language was designed to be simple to use so that a system administrator, or even a user, could easily integrate a new editor. The scripts are in plain text format so they may be edited with any text editor as well as with the DXM's own very simple script editor.

Script Structure

Each script comes in two parts: a configuration file, which sets system parameters such as the location of the editor and the users document directory; and the batch file which contains two sets of commands: those that are executed to load a document into the editor, and those which store the document back to the database. The scripts are structured in this way so that the batch file contains all those commands that will remain the same for the editor on different machines or for different users, whereas the configuration file contains settings that will change. This allows users to share scripts easily.

The logic structure of the script language was kept as simple as possible. There are control instructions, no loops, no If-Then statements, no functions etc. Each script just consists of two lists of instructions - one for loading the document and one for storing it. In future work, this will be extended to include a list for other functions for example for following hyperlinks, but the basic structure will not change, or to provide special sections which may be called from an editor's macro language or API.

Basic Actions

The majority of instructions in the DEScript language are file manipulations. This includes retrieving files from the SIM database, moving files around on the PC's hard disk, adding lines to and deleting lines from files, changing strings in a files etc. Some examples of these commands are:

`AddLine(filename, line_string)`
Adds the string specified as a new line to the end of the specified file.

`DelLineNum(filename, line_number)`
Deletes specified line from specified file

```
FetchFile(type, destination)
StoreFile(type, source_file)
```
These commands fetch a file from a database or store a file to the database.

```
GetEnv(variable_name, return_string)
```
Returns the current value of an environment variable.

User Interaction

Many editors require the user to perform certain actions to load or to save a file. These might include selecting the File menu, Selecting the "Open" option, either selecting a file from a list or just typing in a filename etc. The DEScript language enables the DXM to automate these actions by simply sending the required mouse actions and keystrokes to the editor, thereby simulating the user's actions. This technique is known as "keystuffing", and it essentially uses the power of the operating system to provide a connection between applications.

The DEScript instructions for keystuffing are currently more complex than other kinds of instructions due to the complexities involved in sending keys to Windows applications. An example set of keystuffing instructions follows. This set of instructions automatically starts the editor, selects the "File" menu, selects the "Open" menu item, types the name of the file which is to be edited and then clicks on the "OK" button.

```
SystemSync('$EDITOR_PATH', InContext', ' ', 'Single', '$progID1')
```
Starts the "InContext" editor or uses the current copy if it is already running...

```
WindowAction('$progID1','ALT_KEY',
  FO','[1,#32770,Open], [1,Edit,
  ]' ,'$winHan2')
```
Selects the "File" menu, then the "Open" item using their shortcut keys "F" and "O".

```
WindowAction('$winHandle2','SEND_S
  TRING','$TEMP_DOCNAME',
  1,#32770,InContext],1,Button,OK
  ]','$winHan3')
```
Clicks the "OK" button.

These sorts of actions require the "Window Class Names" to appear in the `"WindowAction"` instructions (**bold**ed). These are identifiers that were allocated by the designer of the editor software at the time that they wrote the program. SIMDAS provides the user with a simple tool for obtaining the class name from an application's window.

4.4 Integration Examples

Figure 3 shows the complete script for integrating the "InContext" SGML editor.

```
C:\SIMDAS\ICCONF.TXT
$DTD_PATH='c:\sim\simpci_c\bin'
$TEMP_DOC_NAME='D:\temp\ic_doc.sgm'

C:\SIMDAS\ICBAT.TXT
[EDIT]
ReplaceBlock('$DOC_NAME', 'das', 'dtd', '${DTD_PATH}\${DTD_NAME}')
SystemSync('$EDITOR_PATH', 'InContext', ' ', '$progID1')
GetWinHandle('$progID1', '$winHandle1')
WindowAction('$winHandle1','ALT_KEY', 'FO','[1,#32770,Open],
      [1,Edit, ]','$winHandle2')
WindowAction('$winHandle2','SEND_STRING','$TEMP_DOC_NAME',
      '[1,#32770,InContext],[1,Button,OK]','$winHandle3')
WindowAction('$winHandle3', 'MOUSE_CLICK','&H10010',
      '[] ','$dummyWinHand')
EditMsgBox('InContext : Press the store button if you want to store
your document to the database')

[STORE]
ReplaceBlock('$DOC_NAME', '${DTD_PATH','${DTD_NAME}',${DTDNAME})
WindowAction('$winHandle1', 'ALT_KEY', 'FA',
      '[1,#32770,Save As],[1,Edit, ]', '$winHandle3')
WindowAction('$winHandle3', 'SEND_STRING', '$DOC_NAME',
      '[1,#32770,Save As], [1,Button,&Yes]','$winHandle4')
#The Post condition for the next command is the change of time stamp
# in the $DOC_NAME
WindowAction('$winHandle4', 'MOUSE_CLICK', '&H10010', '$DOC_NAME',
      '$dummyWinHandle')
```

Figure 3 - Example DEScript Script

4.5 Discussion

The script language approach has allowed us to build a system which enables a user to integrate any SGML editor with the SIM database. This means that as new SGML authoring tools come onto the market, or as the users needs change they can change the editing package they use. It also means that different editors could be used to do different kinds of work - for example, a simple spelling error could be corrected using a standard text editor like "Notepad" or more complex changes could be made using a WYSIWYG editor such as "Framemaker+SGML", "Grif" or "Adept Editor".

We have provided here a discussion of how DEScript is used to connect to very closed editors. There are two disadvantages with integration at this level. Firstly, that the "`WindowAction`" command has quite a complex syntax. SIMDAS however provides a tool to assist the user in writing these scripts. Secondly, when the DXM sends keystrokes and mouse clicks to an application the user will see the results of all intermediate actions such as menus being pulled down and filenames being typed. Though it happens very quickly, this can be distracting for the user.

However, to put these areas of difficulty into perspective, at least SIMDAS can connect to these very closed editors at all, even if the approach is inelegant. Of course the one of the strengths of the SIMDAS is that it is able to make the most of the facilities of more open editors as is discussed below.

5. Making the most of Openness

The major feature of the "lowest common denominator" approach is that it enables us to integrate the SIMPCI with any editor at a basic level. However that if an editor provides a better level of integration then our aim was to connect with the editor at that level. High levels methods of integration which might include OLE communication or an API can allow two tools to be integrated in a very smooth manner which makes them very easy to use in combination, and we wish to make the most of this sort of integration where it is available.

5.1 Microsoft Word - DDE Communication

After being introduced to the SIMDAS project, many of Ferntree's SIM customers commented that they would like to use it store their Microsoft Word documents in the SIM database so that they could use SIM as a document repository.

Word Documents in SIM

Firstly, in order to store Word documents in the SIM database we first require that they are in Microsoft's Rich Text Format (RTF) rather than in Word's native format. (`.doc`) This allows us to store the document as a string rather than as an amorphous binary object. Secondly, in order for RTF documents to be indexed by the SIM database they are stored with the following record structure:

```
RTFRecord
{
  RecordNumber: integer
  RTF:      string
  SGML:     sgml
}
```

The "`SGML`" field is a "virtual field" which means that it does not represent real data, rather it is generated automatically when it is accessed. In this case, the generation is performed by a filter which takes the "Data" field containing the RTF document and transforms it into SGML. This technique enables SIM to index SGML virtual field whilst retaining the RTF document so that it may be viewed and edited from Microsoft Word.

5.2 Planned Development

The work in connecting to editors at the OLE, API and CORBA levels had not been completed at the time of writing. However, following is a discussion of our plans for this development.

OLE and OpenDoc

Many applications which operate under Microsoft Windows use the OLE standard to enable communication with other software. As mentioned previously this does not apply to all SGML editors, but the market appears to be moving towards OLE as a means of communication and integration.

The OLE system enables applications to make software objects publicly available to that other applications may send messages to them or access their data. This can be used in a number of ways, from providing a simple "socket like" interface (as is used between the DXM and the SIMPCI) to embedding software objects to form a compound document. We wish to enable the DEScript language to instruct the DXM how to communicate with another application using the OLE protocol.

[EDIT]
DDESend('WINWORD', 'SYSTEM', '[FileOpen "${DOC_NAME}"]'**)**
EditMsgBox('Click "Store" to store file to "${DB_NAME}" database.'**)**

[STORE]
DDESend('WINWORD', 'SYSTEM', '[FileSave]'**)**

[Browse]
DDESend('WINWORD', 'SYSTEM', '[FileOpen "${DOC_NAME}"]'**)**

Figure 4 - DEScript Script for Word

Connecting to Word

Despite the obvious differences between Microsoft Word and most SGML editors the SIMDAS connection to Word was remarkably simple. In fact, the script for Word is the simplest that we have written, being only 7 lines long as shown in Figure 4.

In this script, Microsoft's DDE protocol is being used in the `[Edit]` and `[Browse]` sections to open the document and then in the `[Store]` section to save the document again. The simplicity of this script in this example shows how we can utilise the "openness" of the Word editor when editing RTF documents.

In order to build OLE communication into DEScript, the DXM needs to make a mapping of the names of an object's methods to the Interface Identifiers (IID's) of those methods. This enables the user to simple type the name of the method into the integration script and for DXM to interpret this make the appropriate call on the OLE object. An example DEScript may then look like this:

```
OLE("Word.Basic.FileOpen(.Name=$DOC_NAME)")
```

This command calls the "`FileOpen`" method on the "`Basic`" object as served by the OLE application Microsoft "`Word`", passing it the string "`.Name=$DOC_NAME`" as a parameter

Integration in this way using the OpenDoc system is currently under investigation.

API Level Integration

Integration at between the DXM and an application using the applications API can occur in two directions. Firstly, the user can use the API to extend the tool by adding functions which make calls to the DXM's library functions. An example of this might be to add a "Store" button to an SGML editor which would then make a call to the DXM and cause it to execute the "Store" section of that editor's DEScript batch file.

The second way in which a tool's API might be used is for the DEScript language to provide facilities which enable the user to include the tool's API functions in the script. An example of this would be where the user put a line in the "Edit" section of an SGML editor's DEScript batch file which calls the editor's "`OpenDocument`" function passing the filename of the document as a parameter.

Example of API Level Integration - Grif

Grif is a WYSIWYG SGML editor, developed and marketed by Grif.sa, a French company. We have not actually made the integration as Grif currently cant run on Windows NT, rather we have investigated the method of the integration based on the Grif documentation.

Grif comes in two complementary version: The Grif editor and the Grif Application Toolkit Environment (GATE) [GATE] that offers an API. With GATE, it is possible to add, remove, disable or redefine menus and editing functions. It is also possible to use application specific code to access and modify part of a document, and to access other applications through bi-directional channels.

For the purpose of close integration with the SIMDAS DXM, we make the following modifications to Grif:

1. Extend the menu with a "Store" function (as we do not want to overload the local "Save" function) The SMDAS store function will store the document back to the database, possibly after having created a new record in case it is a new document.
2. Modify the "Open" function to give the user the option to read a document from the database rather than from the PC's the file system.

These two changes can be made by adding the appropriate messages to the EDITOR.I file. This file, which is in GATE's "I language", enables messages to be associated with elements of the Grif user interface. An example of such interface is the following:

```
APPLICATION EDITOR
```
Section heading...

```
DEFAULT
StdOpenDoc.PRE:  SIMDASOpen;
```
Call the SIMDASOpen" function before the standard Grif "Open" function is called

```
StoreToDatabase:  SIMDASSave;
```
Call the SIMDASSave function when the "StoreInDatabase" message is received. (See below)

```
MENUS
StandardMenus;
Store:  StoreToDatabase
```
Adds a "Store" menu to the interface and associates the "`StoreInDatabase`" message with that menu item.

"`StdOpenDOC`" is a standard Grif message, while "`StoreInDatabase`" is a new message which will be sent when the Store Menu is activated. The user then simply writes two short C functions, namely "`EditorSIMDASOpen`" and "`EditorSIMDASSave`" and links them with Grif . These functions will call the corresponding functions in the SIMDAS DXM library. Parameters like the name of the current document, can be retrieved using Grif API so that they may be passed to the DXM.

The rest of the integration is similar to the one done for closed tools, through the DEScript batch file:

```
[Edit]
SystemSync('$EDITOR_PATH
$DOC_NAME', "Grif", ' ',
'$progId1')

#Set Grif Environment Variables to
values set in config file
SetEnv('RESDIR', '$GRIF_RES_DIR')
```
Sets directory where Grif needs to search the localisation files of the user interface

```
SetEnv('LANG ', '$GRIF_ LANG')
```
Specifies the current language ("en" for English, "fr" for French)

```
SetEnv('GRIFSCH', '$GRIF_
GRIFSCH')
```
specify one or more directories where Grif will search the presentation and DTD

```
SetEnv('GRIFDOC', '$GRIF_
GRIFDOC')
```
specify the directory for documents

```
SetEnv('MDIMODE', '$GRIF_
MDIMODE')
```
specify whether the interface is MDI (multiple display interface) mode or not

```
SetEnv('STARTUP', '$GRIF_
STARTUP')
```
Specify one or more start-up file

6. Conclusion

Through our work in the SIMDAS project we have shown that it is possible to connect completely closed software tools in a way that appears to be very simple from the perspective of the user. This allowed us to connect SGML editors to an SGML database, but the approach could easily used in many other kinds of systems. We think that this result is particularly important for document computing systems for two reasons.

Firstly, users may want to use different tools in their work on a document - an editor, a version control system, a document management system or database etc. Our system provides a mechanism for these kinds of tools (and more) to interact in ways that may not have been foreseen by their designers.

Secondly, it enables different users to integrate the tools which they require. As there are many different kinds of users with different requirements , a great deal of flexibility is required in a document system. Our approach provides this flexibility and also enables new software tools to be integrated into existing systems as they become available.

Although the "Lowest Common Denominator" approach has been very successful, we feel that it does not allows us to make the most of the editors which are more open. In addition to this basic level of integration, we wish the DXM to provide integration at the level of OLE communication and to be called from an editor's API. To do this the DXM will provide the facility to make calls on an OLE object's interfaces and it defines a programming interface which may be used by the macro language which is available to an editor (e.g. WordBasic), or which may be called from an editor's API written in C. Communication between tools at this OLE and API level will provide very flexible integration and is a more "elegant" solution than keystuffing from the user perspective as the opening of windows and clicking of buttons no longer appears to happen automatically before the user's eyes.

We feel that these two approaches combined allows our system to make the most of an editor's openness - it can interoperate at a basic level if it needs to, or make use of more sophisticated facilities if they are available. While this does not replace the need for more global interoperability solutions such as OLE or OpenDoc, it enables us to achieve connectivity between tools in an environment where these connectivity standards have not been universally adopted - and in doing so to create a useful and flexible document management system.

Acknowledgements

We are most grateful to our colleague Mr Chi Tsang who wrote the DXM's script interpreter and perhaps more importantly took on extra work so that we might have time to write this paper.

About the Authors

Brendan Hills is a Software Engineer at the CSIRO Division of Information Technology in Melbourne. His current work is in electronic documents, SGML, and document management. He has been heavily involved for the architecture, design and construction of the SIMDAS document management system.

Anne-Marie Vercoustre is the leader of the Text-based Information Management research group at the CSIRO Division of Information Technology. She is currently on leave from INRIA, (a French government research organisation) where she has been mostly involved in syntax based programming environments, structured documents and hypertext. She participated in the design of Symposia, the first WYSIWYG HTML editor connected to the Web, based on the Grif SGML Editor (Grif.SA). Her current research interest includes reusability of fragments of documents through the construction virtual documents upon SGML databases.

Appendix A - SGML Editors

- Adept (Arbortext)
- Author/Editor (SoftQuad)
- EditTIME (T.I.M.E LUX)
- FrameMaker+SGML (Frame)
- Grif SGML Editor (Grif.SA)
- InContext (XSoft)
- Interactive Authoring and Display System (Public domain)
- NEAR & FAR Author (Microstar)
- SGML Author (Microsoft)
- WordPerfect Intellitag (WordPerfect)
- WriterStation (Datalogics)
    - TagWizard (Nice Technologies)

Appendix B - HTML Editors

HTML editors are included in this discussion for three reasons:

1. As SGML becomes more popular or the limitations of HTML become more important, HTML editors may be developed into SGML editors.
2. Some HTML editors are more open and more user friendly and it is possible that SGML editors will be more like this in the future.
3. If a client uses SIM to store HTML documents they may wish to select an HTML editor rather than a generic SGML editor.

- HotDog (Sausage Software)
- HotMetal (SoftQuad)
- HTML Assistant (Public domain)
- HTML HyperEdit (Public domain)
- HTMLEd (Public domain)
- NaviPress (NaviSoft)
- Netscape Gold Editor (Netscape)
- Symposia (INRIA/Grif.SA)
- WebEdit (Ivory Tower Software - Public domain)

7. References

[BER]
Berners-Lee, T & Connolly, D., 1995, *Hypertext Markup Language Specifications - 2.0, Internet Draft*, `http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_2.html`

[BLUE]
IBM, *BluePrint 1.0*, `http://www.software.ibm.com/openblue/IDOH1.HTM`

[BROC95]
Brockschmidt, K., 1995, *Inside OLE*, second edition, Microsoft Press, USA.

[BURK95]
Burkett, W.C, Yang, Y.W, ,1995, *The STEP Integration Information Architecture, Engineering With Computers*, Vol.11(3), pp.136-144.

[CFI]
CFI Overview : *Enabling CAD Tool Interoperability*, `http://www.cfi.org/cfi/`

[RMIT95:1]
The Structured Information Manager (SIM), `http://www.kbs.RMIT.edu.au/sim/sim.html`

[RMIT95:2]
RMIT, *SIMPCI Users Guide*.

[CLE95]
Cleveland, G., 1995 *Overview of Document Management Technology*, `http://www.nlc-bnc.ca/VI/5/op/udtop2.htm`

[CORBA]
Object Management Group Inc., 1993, *The Common Object Request Broker: Architecture and Specification*, Revision 1.2, OMG Document Number 93.12.43.

[DMA]
DMA: Document Management Alliance `http://www.xerox.com/XSoft/dma.html`

[GATE]
Grif. SA., *GATE: The Grif Application Programming Interface*, Technical Documentation.

[INFO]
Inforium, *LivePage*, `http://www.inforium.com/`

[ISO:1]
ISO, 1995, *ANSI/ISO Z39.50 - Information retrieval: Application Service Definition and Protocol Specification*, Versions 2 and 3.

[ISO:2]
ISO, 1986, *ISO 8879 - Information Processing - Text and Office Systems- Standard Generalised Mark-up Language (SGML).*

[NAV]
Navisoft, *Navipress*, `http://www.navisoft.com/`

[OLE]
Microsoft Corporation, *Object linking and Embedding: OLE 2.0 Design Specification*, 1993.

[OMG]
Object Management Group, 1996, *CORBA: Architecture and Specification*, OMG, USA. (Updated regularly)

[OPENDOC]
The OpenDoc Design Team, 1994, *Introduction to OpenDoc*, `http://www.info.apple.com/dev/du/intro_to_opendoc/iod0_index.html`

[ORFA96]
Orfali, R., Harkey, D. & Edwards, J., 1996, *The Essential Distributed Object Survival Guide*, Wiley, USA.

[PAO95]
Paoli, J., 1995, "Rules for extending a WWW client: The Symposia API", *Proceedings of WWW4*, December 11-14, 1995, Boston, Massachusetts, USA.

[QUI86]
Quint, V. & Vatton, I., 1986, "Grif: An Interactive System for Structured Document Manipulation", Proceedings of *Text Processing and Document Manipulation, van Vilet (ed.), pp 200-213, Cambridge University Press.*

[QUI94]
Quint, V. & Vatton, I., 1994, "Making Structured Documents Active'',*Electronic Publishing--Origination, Dissemination and Design*, vol. 7, num. 1, pp. 55-74,.

[VISBAS]
Microsoft, 1996*Visual Basic Users Guide,* Microsoft Press, USA

[VISC++]
Microsoft, *Visual C++ Users Guide,* Microsoft Press, USA

[WAR95]
Warren-Smith, R.F., *What is Interoperability?*, `http://www.roe.ac.uk/computing/starlink/docs/sgp42.htx/node25.html`