Interoperability of Closed Tools

Building an SGML Authoring System

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

brendan.hills@dit.csiro.au, chi.tsang@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 software tools.

This paper provides a detailed discussion of our solution of the issue of interoperability between closed software tools - SGML editors in particular. The context for this is our current project SIMDAS (SIM Document Authoring System) which aims to integrate 3rd party tools provide a document authoring environment for the SIM SGML database.

Keywords Document management, SGML, Document authoring, Document computing, Open systems, OLE, Windows. Interoperability.

1. Introduction

The Structured General Markup Language (SGML) 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 also, 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 it's use in 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 Markup Language (HTML) - the language of documents on the WWW. It is the use of SGML in the WWW which has given SGML it's greatest exposure and which has sparked an increased interest in building tools, such as databases, editors and document management systems which work with SGML.

Our project is to provide a Windows based document authoring system for Ferntree and CITRI's "Structured Information Manager" SGML document database system (SIM). This system 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, it is likely that they will become increasingly so, partially due to the popularity of hypertext on the WWW. Our intention is to provide interoperability between existing SGML tools rather than to write our own. 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. These tools 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.

Proceedings of the 1st Asutralian Document Computing Symposium,
Melbourne Australia,
March 20-21 1996.

This issue will be important for any software system which attempts to integrate with, extend on, or interact with other 3rd party software systems. Interoperability is particularly relevant to document computing as document standards such as SGML give us a format which allows us to work on the same document with different programs. In order to combine these disparate programs into a single system, we must find a way in which they can interoperate.

This paper provides background on the current project - SIMDAS - and on the SIM database and the SIMPCI (SIM PC Interface) client. It then describes the requirements of the current project. An examination is presented of two possible architectures for SIMDAS - with SIMDAS integrated tightly with the SIMPCI; and with SIMDAS dependant on, but separate to the SIMPCI.

An overview of the SIMDAS prototype implementation is given with a discussion of the issues arising from its evaluation.

2. Background

2.1 SGML - Document Exchange

SGML (Standard Generalised Markup 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 user of 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 desires.

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 closed 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 CITRI research organisation and marketed and sold by the Ferntree Computer Corporation. SIM has a client-server based architecture based on the ANSI Z39.50 client-server protocol with servers running on Sun SPARC workstations running the Solaris operating system and clients running on any machine which has 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 Microsoft Windows, and World Wide Web clients such as Mosaic and Netscape.

2.3 The SIM PC Interface (SIMPCI)

The SIMPCI is a SIM database query client which runs in the Microsoft Windows environment (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., 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 denote 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.

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)
- Can be sold cheaply, perhaps around $100 AUD per user.
- An optional add-on to SIMPCI - to be sold separately.
- 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. For the requirements phase of this project we did not have any input from actual users of the SIMDAS project, instead user requirements were generated by exploring hypothetical use cases. This lack of real user requirements encouraged us to design SIMDAS so that it may be used in a wide variety ways by different kinds of users.

The requirements for the SIMDAS project were:

- 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 shareware and as commercial product, we would not attempt to build our own editor, rather we would build SIMDAS around and existing tool.

4. 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 extension to the highly popular Windows based word processor "Microsoft Word". We felt this would make it a popular choice of SGML editors amongst people who already used 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 separate WYSIWYG view of their documents.

4.1 The Prototype Approach

The SIM development team had already built the Windows based SIM client (SIMPCI) which provides all of the standard database query facilities, and we wished to avoid duplicating this effort. Our project then became the task of providing the software "glue" to connect the SIMPCI client with the TagWizard SGML editor.

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

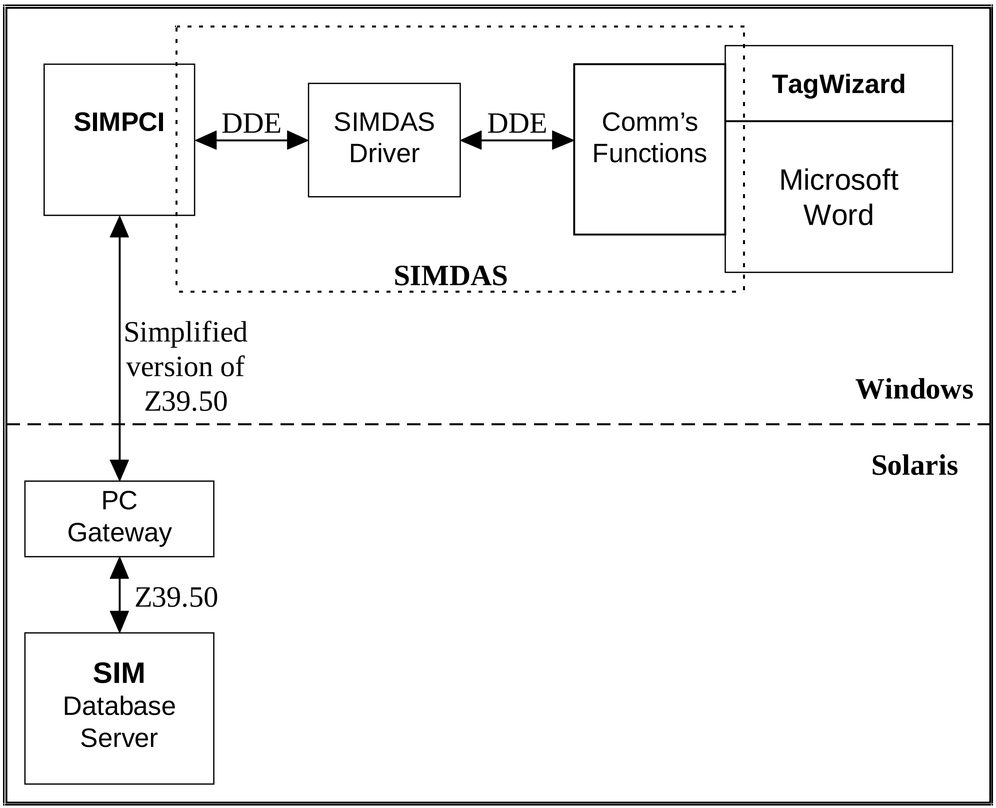

Figure 1 - SIMDAS Prototype Architecture

libraries) and we were therefore able to provide a smooth integration between SIM and TagWizard.

4.2 Discussion

The SIMDAS prototype was successful in that it illustrated 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. Further, it enabled the user to do off-line editing and to follow hyperlinks that were within the edited document and then to edit the document which was the target of the link.

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

Also, during the development of the prototype there were new releases of the TagWizard product, but we were unable to incorporate these later releases due to the work involved in making all the modifications to TagWizard to enable DDE communication with the SIMDAS Driver. This problem is indicative of a flaw in our prototype approach - that we were committed to maintaining our custom modifications to the TagWizard product if we were to keep in step with it's development - we could not be "forward compatible". We could easily have run into a similar problem with the source code modification at the SIMPCI end, but as CITRI was collaborating with CSIRO on the SIMDAS project, CITRI agreed to maintain the DDE communication functionality in future releases of the SIMPCI product.

The other major problem with the SIMDAS prototype was the limitation of being locked into using a single SGML editor - TagWizard. 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.

5. Version 1 - Integrate with any Editor

5.1 Design considerations

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 4.2.

5.2 Version 1 - Approaches

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 SIMDAS project team had spent considerable time understanding and then modifying the source code of TagWizard's Word macros and transferring these modifications to a new version of TagWizard would be almost as time consuming.

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. If we could do this, it would solve the problem of forwards compatibility.

The disadvantage with this solution however is that it still locks the SIM user into a particular SGML editing package, as even if that package is better than TagWizard, it still restricts the user.

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 points made above - as trying remain compatible several different editors would be much more difficult than with just a single editor. With this aim, we looked at many different SGML editors (See section 5.3) 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 very few of the SGML editors on the market actually had public interfaces and, despite the recent attention on "standards" such as OLE, when they did have public interfaces, they tended to differ greatly from each other. We found that the SGML editors on the market had varying degrees of "openness" as follows:

1. Completely closed
2. Command line options
3. Modifiable configuration file (e.g. TagWizard's "SGMLWRKS.INI" file)
4. Macro language (e.g. Microsoft Word's "WordBasic')
5. Source code is available
6. DDE communication
7. OLE communication
8. API is available

Most editors fell into categories 1 or 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

5.3 Editors Surveyed

We surveyed 12 SGML editors and 9 HTML editors to gain a good understanding of the sorts of actions needed for integration with an editor. These editors are list in Appendix A and B.

5.4 Integrating with Closed Editors

Integrating with completely closed editors is 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 survey a wide selection of editors to determine what kinds of things SIMDAS would need to do to integrate with a closed editor. (A summary of the editors that we survey is given in section 5.3.) We then designed SIMDAS to have two separate components -- the Document Exchange Manager, 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.

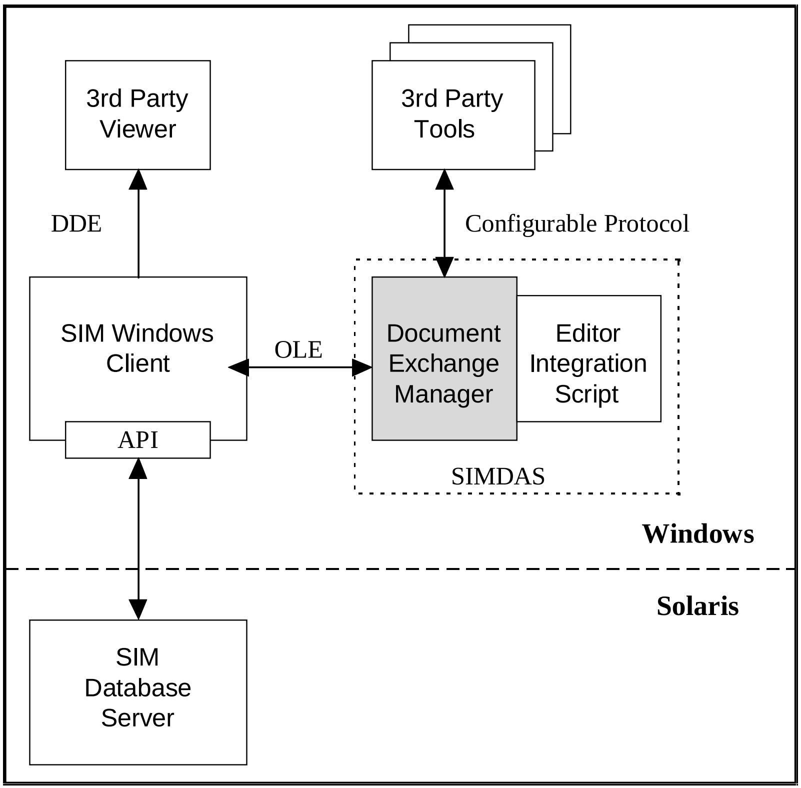

Figure 2 - SIMDAS Version 1 Architecture

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

5.5 DXM - Document Exchange Manager

The DXM performs 4 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 (OLE now being well supported in tools such as Visual Basic) so the transition seemed prudent 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. As described in section 0, 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 or save 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 files which all editors require. These files are stored in the SIM database and a retrieved by the DXM when the editors requires them.

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

5.6 Configuration Scripts

Approach

SIMDAS configuration scripts are written in a purpose-built script language called "DEScript". 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 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, not 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 following hyperlinks and possibly for other functions, but the basic structure will not change.

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".

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('$EDITOR_PATH',
      InContext', ' ', '$progID1')
```
Starts the "InContext" editor...

```
GetWinHandle('$progID1', '$winHan1')
WindowAction('$winHan1','ALT_KEY',
      FO','[1,#32770,Open],
      [1,Edit, ]','$winHan2')
```
Selects the "File" menu, then the "Open" item using their shortcut keys "F" and "O".

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

The problem with these sorts of actions is the need for the "Window Class Names" in the `WindowAction` instructions (**bold**ed). These identifiers were allocated by the designer of the editor software when they were writing the program. They are not easily accessible except with specialised tools such as "Spy++" - a programming tool that comes with Microsoft's "Visual C++" C++ compiler. Clearly most users will not have access to this sort of tool, so we need to either eliminate the need to use of these identifiers, or to provide our own simple tool for finding them. A third approach would be to eliminate the writing of these sorts of script instructions by providing a script recorder which enables the user to generate the script simply by performing the required actions. It is this third path that we intend to follow as described in the following discussion.

5.7 Integration Examples

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

5.8 Discussion

The "lowest common denominator" 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 products 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 spelling error could b corrected using a standard text editor like "Notepad" or more complex changes could be made using "Framemaker+SGML", "Grif" or "Adept".

The biggest limitation with this solution however is the complexity of the scripts when it comes to keystuffing. As a possible solution to this problem we are experimenting with the idea of a 'script recorder" which will generate the appropriate DEScript instructions as the user performs the actions that are required. Once the script has be generated in this way, the script will be used as normal.

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

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.

7. Further Work

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. For example, if an editor provides a mechanism for OLE communication, our current implementation of the DXM is unable to make use of this. Further, if an editor has a public API, as

Framemaker+SGML and Grif do, we would like the DXM to be able to communicate at that level. We call this "Making the most of Openness", and our current work is to extend the approach used in Version 1 to provide configurable integration at these higher levels[1].

The DXM 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 or Visual Basic. The system administrator is just required to write a few defined functions for each new editor using the editors own macro language or API.

Communication between tools at this 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.

[1] Watch out for a forthcoming paper on this work!

Appendix A - SGML Editors

Following is a review of SGML editors including a description of their degree of openness and of their available means of communication.

- **Adept (Arbortext)**

- **Author/Editor (SoftQuad)**

- **EditTIME (T.I.M.E LUX)**

- **FrameMaker+SGML (Frame)**

- **Grif SGML Editor (Grif.SA)**

Grif is a WYSIWYG SGML editor. The editor itself is completely closed but there is an API - Grif Application Toolkit Environment (GATE) - which allows the programmer to extend Grif's menus and to build documents from external applications. GATE is expensive.

- **InContext (XSoft)**

InContext is a non-WYSIWYG editor.

- **Interactive Authoring and Display System (Public domain)**

- **NEAR & FAR Author (Microstar)**

- **SGML Author (Microsoft)**

Microsoft SGML Author is an ad-on macro package for Microsoft Word.

- **TagWizard (Nice Technologies)**

TagWizard provides macro's and a DLL which provide SGML editing facilities to Microsoft Word. TagWizard's functions may be called from the WordBasic macro language, however, TagWizard hides the interfaces to it's functions so it is not possible to access TagWizard in an ad hoc way. As we have the source code for TagWizard version 1.5, we could find a set of necessary functions from the TagWizard macro's and publish the interfaces to these functions to allow SIMDAS system administrators to connect TagWizard with SIMDAS.

- **WordPerfect Intellitag (WordPerfect)**

- **WriterStation (Datalogics)**

Appendix B - HTML Editors

Following is a review of HTML editors including a description of their degree of openness and of their available means of communication. HTML editors are included in this discussion for three reasons:

- As SGML becomes more popular or the limitations of HTML become more important, HTML editors may be developed into SGML editors.
- 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.
- 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)**